2015-09

Investigation of coordination algorithms for swarm robotics conducting area search

Lau, Dylan Z.

Monterey, California: Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

INVESTIGATION OF COORDINATION ALGORITHMS FOR SWARM ROBOTICS CONDUCTING AREA SEARCH

by

Dylan Z. Lau

September 2015

Thesis Co-Advisors: Timothy H. Chung
Duane Davis

This thesis was performed at the MOVES Institute
Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	09-25-2015	Master's Thesis 11-21-2014 to 09-25-2015

4. TITLE AND SUBTITLE

INVESTIGATION OF COORDINATION ALGORITHMS FOR SWARM ROBOTICS CONDUCTING AREA SEARCH

5. FUNDING NUMBERS

6. AUTHOR(S)

Lau, Dylan Z.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

This master's thesis developed an operational coordination controller for enabling sensor-based coverage algorithms for large-scale swarm unmanned aerial vehicles (UAVs). The prototype controller was validated by live-fly field experiments of swarm UAVs carried out in Naval Postgraduate School's (NPS's) field laboratory at Camp Roberts.

The proliferation of hobbyist drones, or UAVs, in recent times has presented the opportunity to field large-scale swarm UAVs. A coordinated swarm of UAVs can prove useful in a surveillance or search mission by rapidly covering the area through data collection from multiple vantage points simultaneously. This thesis designed and implemented an onboard swarm search controller that coordinates the swarm for an autonomous area search with different coverage algorithms. The coverage algorithms are evaluated with simulation and validated by live-fly field experiments.

14. SUBJECT TERMS	15. NUMBER OF PAGES
aerial surveillance, collective robotics, swarm intelligence, swarm search	117
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

INVESTIGATION OF COORDINATION ALGORITHMS FOR SWARM ROBOTICS CONDUCTING AREA SEARCH

Dylan Z. Lau
Civilian, Defence Science and Technology Agency, Singapore
B.Eng., Nanyang Technological University, 2007

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MODELING, VIRTUAL ENVIRONMENTS, AND SIMULATION

from the

NAVAL POSTGRADUATE SCHOOL
September 2015

Author: Dylan Z. Lau

Approved by: Timothy H. Chung
Thesis Co-Advisor

Duane Davis
Thesis Co-Advisor

Christian Darken
Chair, MOVES Academic Committee

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This master's thesis developed an operational coordination controller for enabling sensor-based coverage algorithms for large-scale swarm unmanned aerial vehicles (UAVs). The prototype controller was validated by live-fly field experiments of swarm UAVs carried out in Naval Postgraduate School's (NPS's) field laboratory at Camp Roberts.

The proliferation of hobbyist drones, or UAVs, in recent times has presented the opportunity to field large-scale swarm UAVs. A coordinated swarm of UAVs can prove useful in a surveillance or search mission by rapidly covering the area through data collection from multiple vantage points simultaneously. This thesis designed and implemented an onboard swarm search controller that coordinates the swarm for an autonomous area search with different coverage algorithms. The coverage algorithms are evaluated with simulation and validated by live-fly field experiments.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

ARM	Advanced RISC Machines
ARSENL	Advanced Robotic Systems Engineering Laboratory
DOD	Department of Defense
FPV	first-person view
GPS	Global Positioning System
MOVES	Modeling, Virtual Environments and Simulation
NPS	Naval Postgraduate School
ROS	Robot Operating System
UAV	unmanned aerial vehicle
US	United States
USG	United States government

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

The technological advances brought by revolutions in, for example, the smartphone, have paved the way for the proliferation of hobbyist unmanned aerial vehicles (UAVs). Further, the accessibility of hobbyist UAVs in recent years brings the opportunity for researchers to deploy swarm UAVs. Swarm UAVs have the potential to improve current surveillance or search missions by collecting data from multiple vantage points simultaneously. Researchers have been steadily increasing the swarm size and complexity of the swarm UAVs they are studying. However, autonomous outdoor swarm behaviors are still restricted to formation flying or "flocking." This thesis seeks to advance the field by coordinating an autonomous outdoor swarm of UAVs for an area search.

Theoretical work has identified that the key challenges to coordinating swarm UAVs for area searches include preventing the swarm UAVs from overlapping with each other during the search, and also ensuring that all of the search area is covered. Several approaches are discussed to overcome these two challenges. The following approaches are found to be most suitable for the thesis to implement. To prevent overlaps, a global search map of all swarm UAV positions is maintained through sharing of positional information using a wireless link. A centralized master searcher uses the global search map to decide the search path for itself and other swarm UAVs. The coverage of the search area is tracked by discretizing the search area.

For this thesis, a new swarm search controller is designed and programmed to coordinate swarm UAVs for an area search. The swarm search controller runs onboard the ARSENL fixed-wing swarm UAV (Ritewing Zephyr II). The swarm search controller's role in the ARSENL UAV System Architecture is explained, including details about the controller's software design. The swarm search controller is demonstrated to successfully coordinate area search missions during live-fly field experiments and validates the thesis' approaches.

Also, two coverage algorithms are implemented in the swarm search controller and validated during the live-fly experiments. The first coverage algorithm is a simple greedy algorithm that assigns discretized search cells to swarm UAVs based on closest distance. The second coverage algorithm is a fixed lane algorithm that minimizes overlaps by pre-

allocating discretized search cells to swarm UAVs. For this study, 150 simulation runs are made using both coverage algorithms with different numbers of searchers. The results from the simulation runs are analyzed for statistical significance, and future live-fly experiments are planned to validate the simulation findings.

Analysis of the simulation results reveals that having more searchers does not guarantee a shorter time for complete coverage. The relative position of the searchers' starting position to the search area and the orientation of the search area can have a significant impact to the time for complete coverage. The greedy coverage algorithm has more consistent results when more searchers are involved in the search, while having fewer searchers for the fixed lane coverage algorithm gives more consistent results. Overall, the fixed lane coverage algorithm usually provides a shorter time for complete coverage than the greedy coverage algorithm.

The thesis has shown that research in the swarm UAV field is no longer about realizing the swarm UAVs. The field has matured enough that researchers can shift their focus to pursuing actual real-world applications for swarm UAVs. The live-fly validated swarm search controller represents the first steps in advancing this new focus. By experimenting with different coverage algorithms in the swarm search controller, the thesis opens up a new paradigm by demonstrating that swarm UAVs are ready for software experimentation. This research will undoubtedly be joined by many new and exciting contributions from future researchers.

Acknowledgments

I would like to thank Professor Timothy H. Chung, my advisor, for welcoming me into the Advanced Robotic Systems Engineering Laboratory (ARSENL) team, a positive and high-performance project team he created, and my co-advisor Professor Duane Davis for patiently guiding me. My gratefulness also goes to the Singapore Defense Science and Technology Agency for providing me with the opportunity to pursue my academic interest at this world-class institution.

I would also like to thank my teammates in ARSENL for advice, guidance, and help, especially Michael Clement, Marianna Jones, and Michael Day.

Also deserving thanks are all of my professors and great friends in the Modeling, Virtual Environments and Simulation (MOVES) Institute, who have taught and supported me throughout my stay at Naval Postgraduate School (NPS).

Of course, my thanks go to my parents and family, for sharing their love and encouragement, even if it meant me moving so far away. More than anything I would like to thank my wife, Mei Lu, for the unconditional support, patience, and love she has always provided, and the promising future we have dedicated to each other with our son, Lucas.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

The smartphone revolution has brought about rapid technological advances in electronic miniaturization and battery capacity [1]. These technological advances have paved the way for the proliferation of hobbyist drones, or unmanned aerial vehicles (UAVs). Going for no more than a few thousand dollars each, these UAVs provide autonomous Global Positioning System (GPS) and waypoint navigation capability to anyone without the need for formal training or infrastructure [2]. The reduced costs and increased capability of these UAVs have created an opportunity to deploy large groups (swarms) of UAVs to cooperatively achieve a common goal (or goals). A potential use of a swarm of UAVs is a surveillance or search mission where they can be coordinated to collect data from multiple vantage points simultaneously. Theoretical work identifies the key challenges to coordinating swarm UAVs for area searches. Criteria for evaluating area searches are discussed.

1.1 Rise of Hobbyist UAVs

A UAV is an aircraft that has no pilot onboard. "It can be remotely controlled by a pilot at a ground control station or fly autonomously based on pre-programmed flight plans or more complex dynamic automation systems" [3]. Autonomous UAVs used to be expensive, and only militaries could afford to fly them. Now, people interested in flying an autonomous UAV can easily purchase them from any of the major online retailers for less than a few thousand dollars [2]. Bloomberg reported in 2014 that "Atlanta Hobby, one of the largest independent suppliers of civilian drones in the United States, has seen business jump to about $20 million in annual sales, a 10-fold increase from five years ago" [4]. The sales of these hobbyist UAVs have been growing at a very fast rate.

This surge in hobbyist UAVs in recent years can be attributed to the smartphone revolution. The economies of scale and technological transformation brought by the smartphone revolution has lowered the prices of autonomous UAVs so much that anyone can fly them as a hobby today [1]. The smartphone revolution arguably started in 2007, when Apple introduced the iPhone to the world [5]. Through the years since 2007, many technolog-

ical advances that stemmed from meeting the demands of smartphone users have helped to make the hobbyist UAV a reality. For a UAV to fly autonomously, it needs to have an autopilot, an onboard microprocessor that steers the aircraft based on inputs from the sensors onboard. The traditional desktop processor uses an amount of power that only large military-grade UAVs can afford to carry. To meet the increasing computing needs of smartphone applications while meeting the power limitations of a small mobile device, microprocessor designers such as Advanced RISC Machines (ARM) made developments to their hyper-efficient "reduced instruction set computing" architecture. This type of architecture reduces cost, heat, and power use, making them very suitable for smartphones and hobbyist UAVs. [6]. In 2010, 95% of the processors used in smartphones were ARM-based processors [7].

The sensors required by the autopilot are also found in smartphones, as shown in Table 1.1.

Sensors	**Function**
Gyroscopes	Measure rates of rotation and provide orientation information
Magnetometers	Detect magnetic fields and function as digital compasses
Barometers	Measure atmospheric pressure to calculate altitude
Accelerometers	Measure the force of gravity
GPS	Provides location and time information

Table 1.1: Function of autopilot sensors found in smartphones, after [8], [9]

Because of their use in the smartphone, sensors have also become smaller and cheaper [1]. Other major components of the hobbyist UAV have undergone major transformations. The wireless radio modules have come to embrace 2.4GHz technology, resulting in shorter antennas [10]. Lithium-based batteries have all but replaced nickel cadmium (NiCad) and nickel metal hydrid (NiMH) batteries, due to their higher energy density [11].

In a way, the hobbyist UAV is just a flying smartphone. The economies of scale and technological advances brought about by the smartphone revolution are what made the rise of hobbyist UAVs possible today. Other than hobbyist UAVs, there are also other types of UAVs in the world. Figure 1.1 from [2] shows the different types of UAV and compares the availability of the UAVs.

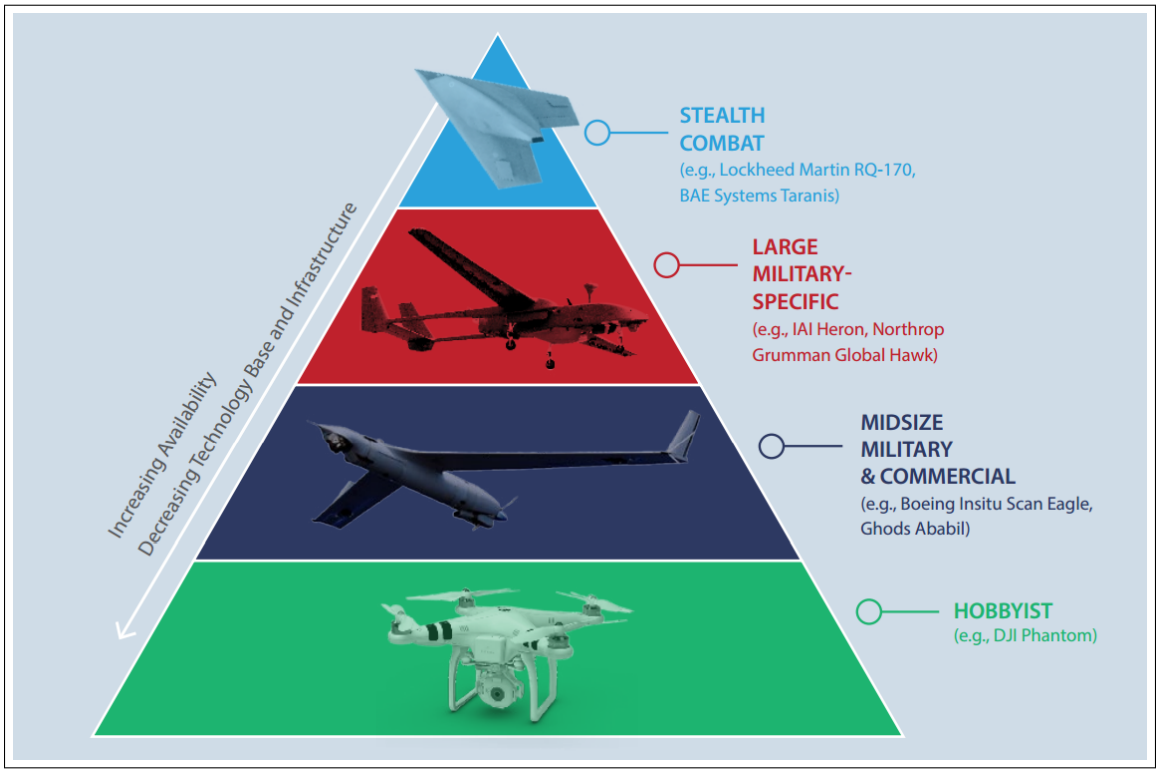

Figure 1.1: Types of UAV and their availability, from [2]

A hobbyist UAV, such as the "DJI Phantom" shown in Figure 1.1, comes with an integrated 1080p video camera and live first-person view (FPV) Wi-Fi streaming of video and telemetry; its cost was about US$700 at the time this thesis was written [12].

1.2 Multi-Vehicle Swarming of UAVs

As hobbyist UAVs became more accessible, potential applications of swarming those UAVs started to surface. An article published in 2010 titled "Towards Autonomous Micro UAV Swarms" suggested that the following application scenarios could be supported by UAV swarms [13]:

- patrol of harbor or borders
- inspection of inaccessible areas such as natural disaster sites, dams, or electric lines
- search phase of a search and rescue mission
- dangerous jobs such as mine detection or fire fighting

- pollution control and environment monitoring
- police video surveillance

The potential applications are huge despite the hobbyist UAVs having a relatively small payload and short endurance compared to the bigger military and commercial UAVs shown in Figure 1.1. The hobbyist UAV cannot see as well or fly as long as its bigger cousins. However, these limitations can be mitigated when the UAVs are deployed in a swarm, as seen in Table 1.2.

Limitation	**Mitigation**
Lower quality sensor	More sensors at different locations
Shorter flight endurance	Team-based recharging schemes [14]
Shorter communication range	Meshed communication networks [15]

Table 1.2: Mitigation of small payload limitations by swarming

In 2001, researchers from the University of the West of England, United Kingdom, used three helium balloons to create a group of flying robots that used infrared sensors to position themselves with each other [16]. The main drawback of using the helium balloons was the severe weight limitation. The researchers could only afford to attach small, lightweight fan units on the balloons that could provide only enough thrust to propel the balloons indoors where the air currents were low.

With the technological advances from the smartphone revolution described in Section 1.1, researchers from Ecole Polytechnique Fédérale de Lausanne, Switzerland, deployed a swarm of 10 autonomous fixed-wing UAVs outdoors in 2011 [17]. The swarm UAVs flew in formation by making use of GPS receivers and wireless modules to share positional information with each other. The swarm UAVs had to avoid collisions with each other, however, by flying at different preassigned flight altitudes.

In an article published in 2013, researchers from the University of Pennsylvania described how they implemented and operated a swarm of 20 micro quadrotors with precision control in an indoor testbed [18]. The quadrotors were able to avoid collisions with processing from a central ground station. The researchers concluded in the paper that “while our quadrotors rely on an external localization system for position estimation and, therefore,

cannot be truly decentralized at this stage, these results represent the first step toward the development of a swarm of micro quadrotors."

In the indoor testbed, a Vicon motion capture system [19] was used to sense the position of each UAV at 100Hz, as shown in Figure 1.2 [18]. The position data was sent to a desktop computer where high-level control and planning was done in a MATLAB environment, and commands were sent to each quadrotor at 100Hz.

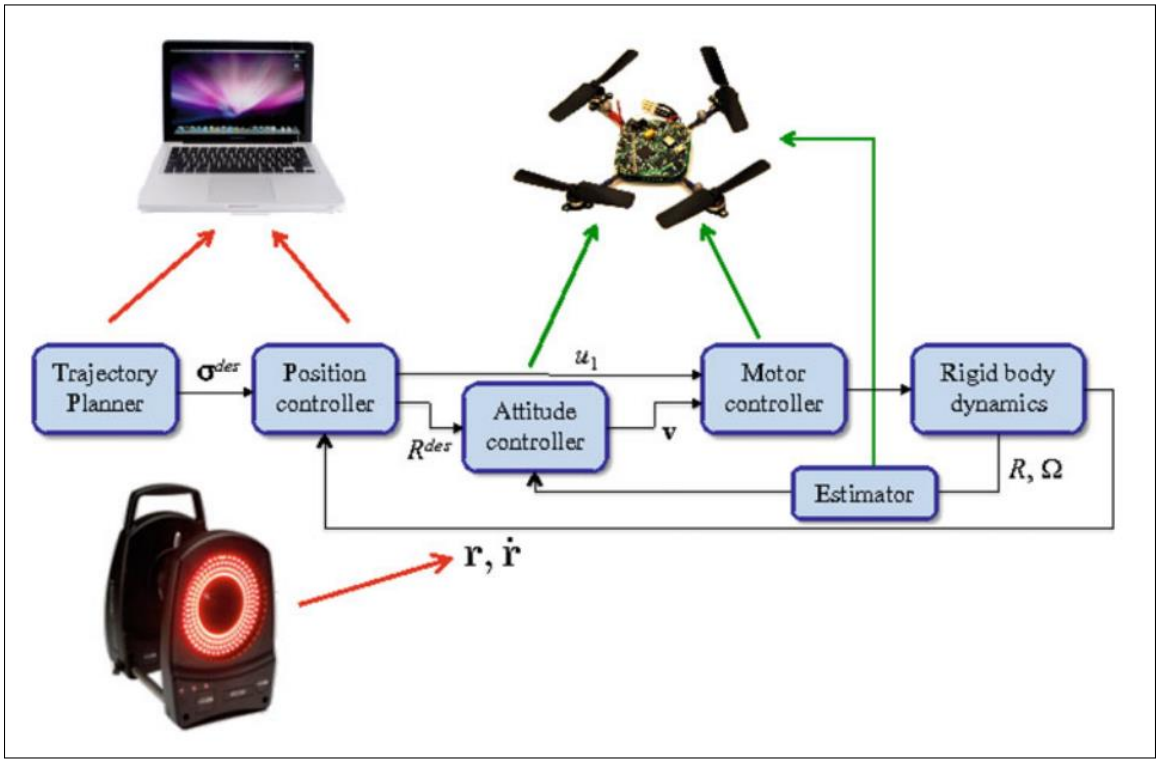

Figure 1.2: Vicon motion capture system in 20 micro quadrotors flight, from [18]

More recently, in a paper published in September 2014, researchers from Eötvös University, Budapest, Hungary, presented a decentralized multi-copter flock that performed stable autonomous outdoor flight with 10 flying agents [20]. Instead of a central ground station that calculates navigational instructions for UAVs in the swarm, the UAVs were able to fly in formation and avoid collisions using onboard processing.

Along with the increase in swarm UAV capability, the size of swarm UAVs has also increased significantly. In August 2015, the team of faculty and researchers from Naval Post-

graduate School (NPS) Advanced Robotic Systems Engineering Laboratory (ARSENL) scaled up and flew a swarm of 50 autonomous fixed-wing UAVs at Camp Roberts, CA [21]. Table 1.3 adapted from [20] compares the various implementations of swarming UAVs in the past few years.

Authors	**Year**	**Vehicle**	**N**	**Decentralized?**	**Collision-avoidance?**	**Terrain**	**Dependency**	**Uniqueness**
Welsby et al. [16]	2001	helium balloon	3	yes	no	indoor	arena	first 3D, relative IR positioning
Hauert et al. [17]	2011	fixed-wing	10	yes	weak / not crucial	outdoor	GPS	first 10 autonomous
Bürkle et al. [13]	2011	quadrotor	5	no interactions	no	outdoor	GPS, ground control station	extendible framework
Hoffmann et al. [22]	2011	quadrotor	3	yes	yes	indoor/ outdoor	GPS+base station outdoor, over-head camera indoor	extendible framework, nice vehicle dynamics
Kushleyev et al. [18]	2012	quadrotor	20	no	not applicable	indoor	VICON, central computer	20 units, best precision control
Turpin et al. [23]	2012	quadrotor	4	SW yes, HW no	yes	indoor	VICON, central computer	quick dynamics
Stirling et al. [24]	2012	quadrotor	3	yes	not applicable	indoor	ferromagnetic ceiling	relative positioning
Quintero et al. [25]	2013	fixed-wing	3	no	no	outdoor	GPS, ground control station	distributed sensing
Vasarhelyi et al. [20]	2014	quadrotor	10	yes	yes	outdoor	GPS	robust outdoor flocking algorithm
Chung et al. [21]	2015	fixed-wing	50	yes	no	outdoor	GPS	first 50 fixed-wing autonomous

Table 1.3: Comparison of different implementations of swarming UAVs, after [20]

From Table 1.3, the most popular vehicle for swarming UAVs is the quadrotor (rotary-wing). With the ability for vertical takeoff and landing, rotary-wing UAVs do not require a runway or launcher that fixed-wing UAVs require. Moreover, only rotary-wing UAVs can hover in place and have the ability to operate in a tight indoor environment. On the flip side, fixed-wing UAVs have more efficient aerodynamics that provide higher speeds, thus enabling a larger survey area per given flight. A fixed-wing UAV swarm is more suitable for area surveillance applications while the rotary-wing UAV swarm is more suitable for indoor or inspection application where precision maneuvering is required. The comparison between fixed-wing UAVs and rotary-wing UAVs is shown in Table 1.4.

UAV Type	**Rotary-wing UAVs**	**Fixed-wing UAVs**
Feature	Can hover	Efficient aerodynamics
Ability	Operate in tight indoor environment	Fly at higher speed (larger survey area per given flight)
Application	Indoor or inspection	Area surveillance

Table 1.4: Comparison between rotary-wing UAVs and fixed-wing UAVs

1.3 Coordinating a Swarm of UAVs for the Perfect Area Search

Building on the previous work listed in Section 1.2, this thesis seeks to develop swarm UAVs towards actual real-world applications. A potential popular application for swarming UAVs is a surveillance or search mission where they can be coordinated to collect data from multiple vantage points simultaneously. Flying swarm UAVs in large formations or "flocking," where individual UAVs negotiate and coordinate their positions in the swarm, has been achieved [20]. What can be investigated further is the swarm's coordination for a collective surveillance or search mission.

Given a fixed area A, the following can be assumed about the perfect area search [26]:

- There is no search overlap (every point in A is searched once before any point is searched twice).
- There is no search conducted outside area A.
- All of area A is covered by the sensor.

Having no overlap and coverage outside the area ensures that no search effort is wasted. No gaps in search coverage guarantees compete coverage of the area.

The sensor used for the area search can be assumed to be a *cookie-cutter*, or sometimes called a *definite range law*, type with range R. This sensor always detects everything within a specified range and never detects anything outside that range [26]:

$$P(detection) = \begin{cases} 1, range \leq R \\ 0, range > R \end{cases} \tag{1.1}$$

A single UAV equipped with the cookie-cutter sensor can conduct the perfect area search by using parallel sweeps (mowing the lawn), spiral-in, or spiral-out paths [27]. The paths prevent the overlap of one searched segment with another, and no search effort is placed outside the search area. The paths can be visualized in Figure 1.3.

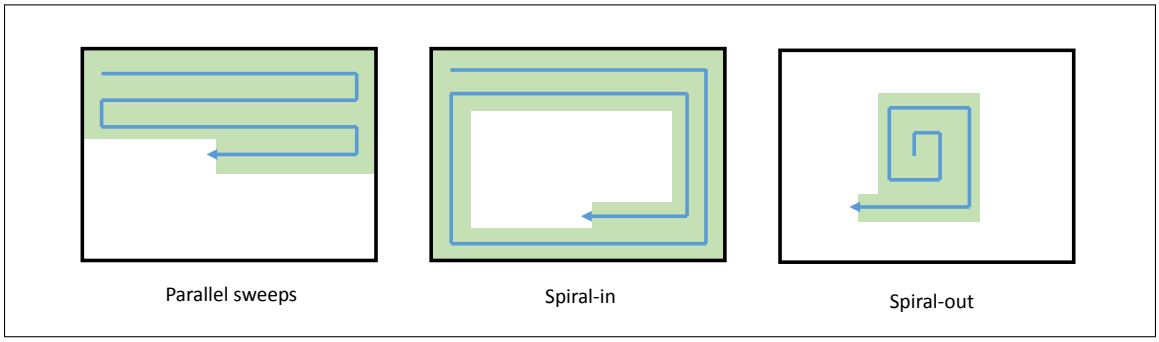

Figure 1.3: Perfect area search paths

The resulting time for the single UAV to complete the perfect area search can be calculated with the following formula [27]:

Notations:

- sweep width = $W = 2R$ (distance)
- search speed = V (distance/time)
- sweep rate = VW (area/time)

$$T = \frac{A}{VW} \tag{1.2}$$

Essentially, the time T to complete the area search is the area divided by the sweep rate. The sweep rate is the search speed V of the UAV multiplied by the sweep width W, which is determined by the range R of the cookie-cutter sensor.

Assuming all the UAVs in a swarm have the same search speed and sensor range, the formula can be modified to calculate the time for a UAV swarm with N number of UAVs to complete the perfect area search.

Additional notations:

- number of UAVs = N (integer)

$$T = \frac{A}{VWN} \tag{1.3}$$

The search area is divided equally among the swarm UAVs to have the best or smallest search time.

So the problem or challenge in coordinating a swarm of UAVs for a perfect area search is to make all the UAVs search the area in such a way that they do not overlap with each other while ensuring that all of the search area will be searched. This problem is divided into two parts, and the approach to resolve them is further explained in their respective sections:

1. The first part is ensuring that all of the area has been searched through discretization of the search area (Section 2.1).
2. The second part is the prevention of overlaps through the maintenance of a "global search map" (Section 2.2).

1.4 Evaluating Swarm UAVs Coverage Algorithms

It can be assumed from Section 1.3 that there is no single straightforward way to conduct a perfect area search. There will be many different area coverage algorithm attempts to be as close as possible to the perfect area search. It does not matter if the algorithm is for a single UAV or a swarm of UAVs; there are two key metrics used to evaluate those area coverage algorithms:

1. Time for complete coverage (How long does it takes for all points in the search area

to be swept at least once?)
2. Missed coverage (How much of the search area is not covered by the searcher?)

An area coverage algorithm is considered superior to another algorithm if it performs better in both of the key metrics. If it performs better in only one of the key metrics, any superiority claim is debatable and will depend on the purpose of the search.

For swarm UAV searches, the following additional metrics may be used to evaluate them:

- Load balancing (What is the distribution of the search area among the swarm UAVs?)
- Communication requirements (How much network traffic is required for the algorithm to work?)
- Communication robustness (How much degradation of the network traffic is possible before the algorithm fails to work?)
- Computational requirements (What is the cyclomatic complexity of the algorithm?)

Implementation of the coverage algorithms in the swarm UAV controller is described in Section 2.5. The description of the evaluating metrics available in the simulation is in Section 2.6.

1.5 Main Contributions

This thesis demonstrates a way for large-scale swarm UAVs to coordinate among themselves to conduct autonomous area search. It paves the way for swarm UAVs to perform practical applications such as police surveillance, cross-border and harbor patrol, and search and rescue. The swarm UAV controller developed for the thesis also allows different coverage algorithms to be evaluated in simulation; it helps to assess and compare potential coverage algorithms. This saves development effort as not all algorithms need to go through field testing.

1.6 Organization of This Thesis

This thesis is divided into five chapters:

- Chapter 1 provides the reason for the proliferation of hobbyist UAVs, the motivation

for swarming UAVs, and the technical challenges of coordinating swarm UAVs for area search.

- Chapter 2 describes the approach to overcoming these technical challenges.
- Chapter 3 presents the results of the approach in simulation and live-fly field experiments.
- Chapter 4 explains the insights and findings from the results.
- Chapter 5 concludes the thesis and proposes future work to build upon the thesis.

CHAPTER 2:
Approach

The primary objective of this thesis is implementing the onboard coordination controller on swarm UAVs for an autonomous live-fly area search. Successful implementation of the coordination architecture is the research goal. The secondary objective of this thesis is evaluating coverage algorithms for performing the area search. This chapter includes explanations for the approach taken as well as the assumptions, designs, and evaluation metrics.

2.1 Discretized Search Area

As a searcher sweeps along the search area, the area behind the searcher's sweep width is assumed to be covered. This can be visualized in Figure 2.1. By placing discrete "search cells" to cover the whole search area, the searcher can be made to sweep over the search area, as seen in Figure 2.2.

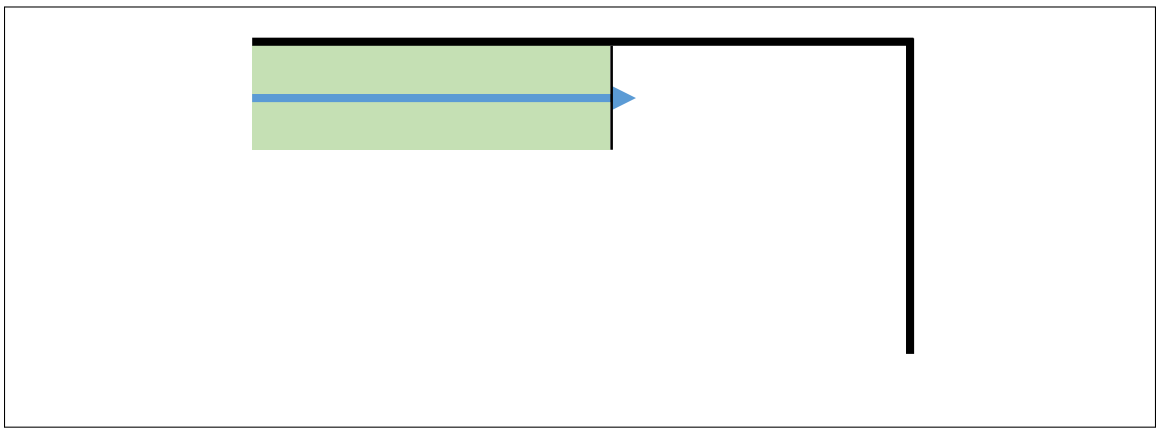

Figure 2.1: Searcher sweeping along the search area

Depending on the interval between the search cells, the search area can be considered to be fully covered when all the search cells are swept over. The interval between the search cells has to be greater or equal to the sweep width to prevent any overlaps and yet as small as possible to prevent any gaps. Choosing the sweep width as the search cells interval causes

gaps to appear in the corner when the UAV turns. To prevent gaps when turning, the sweep width has to be greater than the interval by $\sqrt{2}$. In other words, the interval is $\frac{1}{\sqrt{2}} \approx 70\%$ of the sweep width. Figure 2.3 shows the difference between the two approaches for choosing the search cells interval.

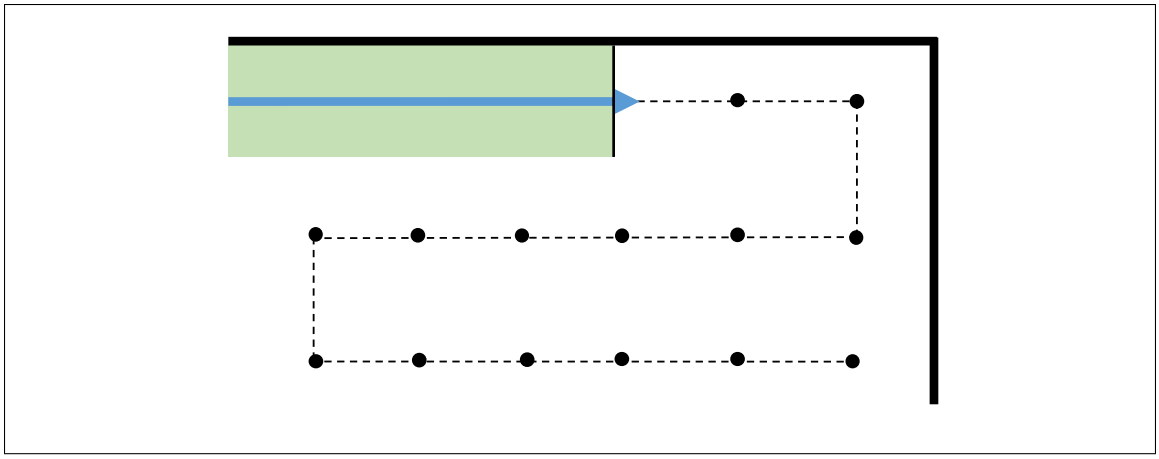

Figure 2.2: Searcher sweeping along the discretized search area

Unless the search area does not require any of the UAVs to turn, there is no way to conduct the perfect area search where there are absolutely no overlaps and gaps in the search. Between the two approaches in Figure 2.3, using the search width as the search cells interval will leave gaps whenever the UAV turns. The other approach will take around 30% more time than the previous approach. For this thesis, the search area is assumed to be completed when all the search cells are swept over by at least one UAV in the swarm.

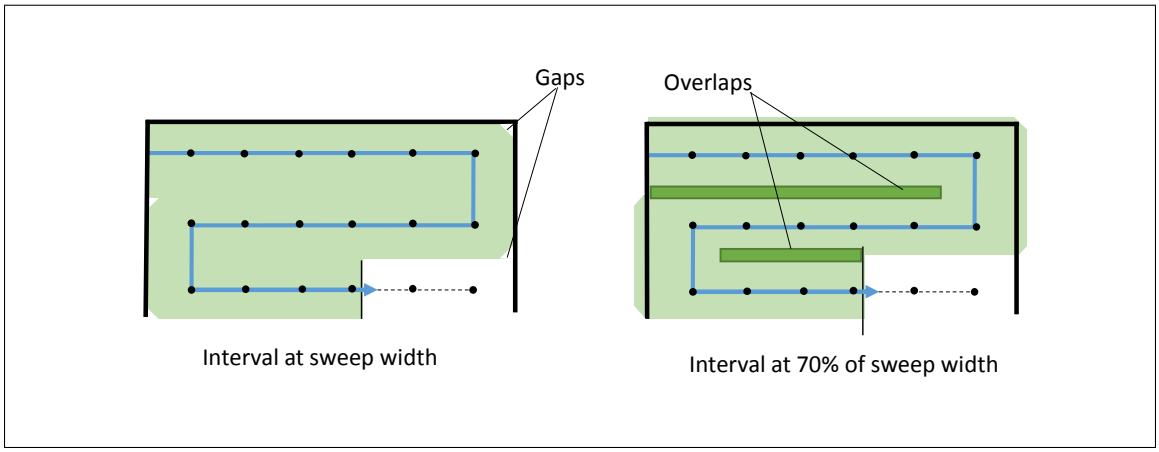

Figure 2.3: Difference between two approaches for search cells interval

2.2 Maintaining the Global Search Map

It is relatively straightforward for a single UAV to keep track of what it has searched. With GPS, it knows where it has been and where it still needs to go without overlaps. For a swarm of UAVs, this information resides in the individual UAVs' memories. Somehow the information needs to be shared among all members of the swarm so, collectively, they know where the swarm has been and where they need to go.

The simplest method is for all the UAVs to broadcast their positions to every other UAV via a wireless link, enabling each receiving UAV to maintain its own copy of the global search map. The quality of the global search map is vital to preventing overlaps. An outdated global search map causes a UAV to move into a position that may have already been searched. The perfect global search map will be one that takes no time to update across all members of the swarm. In reality, this is not possible because the radio module onboard the UAV needs time to encode and decode the position information on the radio frequency. As the number of UAVs increase, the latency for the global search map to get updated will increase as well. The risk of an inconsistent global search map may occur as the network links between the UAVs deteriorate due to the growing number of UAVs.

Another method to maintain the global search map is to break the map down and divide the search area into multiple hierarchical levels of representation. This "divide and conquer" approach is more scalable than the flat representation used in the earlier method. So rather than sharing information from one end of the search area to the other end, the search area

is divided into regions where smaller groups of UAVs maintain a "regional" search map of its region. The visual difference between these two methods can be seen in Figure 2.4.

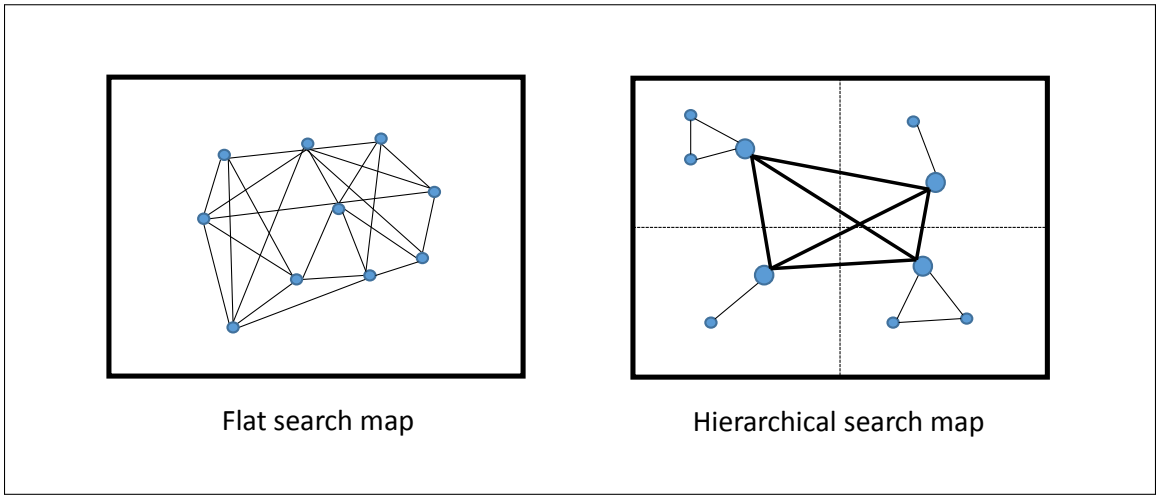

Figure 2.4: Difference approach to maintaining the global search map

The search area is divided into four regions in Figure 2.4. Within each region, a group of UAVs is under the control of a "master searcher," which coordinates the boundary of the regions with other master searchers. The master searcher and the other "slave searchers" only need to broadcast their positions to UAVs in the same region to maintain the regional search map. The advantages and disadvantages of the two methods are summarized in Table 2.1.

With a reliable search map, the next issue for the searchers to decide is where to go next. Assuming the search map is 100% consistent across all searchers, each searcher can run an algorithm to select its next destination and use the same algorithm to predict other searchers' destinations as well. If the search map is not the same across the entire swarm, the searchers may make the wrong decision and cause overlaps. Given the potential of unreliable search maps, the master searcher can also take on the role of an arbiter at a search region. The master searcher receives positional information from all searchers and maintains a single centralized regional search map. The master searcher can make the decision of where each searcher (including itself) has to go without fear of conflicts. Another advantage of this centralized approach is that it is easier to debug if something goes wrong; there

Methods	Advantages	Disadvantages
Single flat global search map	• Resilient to failure of any UAV • Simple to implement	• Network performance limits scalability
Multiple hierarchical regional search maps	• Scalable to large search area	• Failure of a master searcher affects whole region of searchers • Additional complication in coordination efforts between master searchers

Table 2.1: Advantages and disadvantages of flat (global) and hierarchical (regional) search map

is only one place to look. The slave searchers can deploy with cheaper units as they require less memory and processing power. The main disadvantage is that the master searcher becomes a single point of failure, as with the case of the hierarchical search map.

The advantages and disadvantages of centralized and decentralized decision making are summarized in Table 2.2.

There are advantages and disadvantages to each of the approaches in maintaining the global search map. The choice is largely dependent on the scenario of the area search. Table 2.3 shows the recommendations for which search map approach to use depending on the scenario requirements.

For this thesis, the goal is to coordinate a swarm UAV to conduct an area search as quickly as possible with minimum overlaps. A centralized decision-making, single flat global search map approach is selected for the swarm UAV controller.

Methods	Advantages	Disadvantages
Decentralized decision making	• Resilient to failure of any single UAV • Scalable to very large number of searchers	• Additional complication in coordination efforts between searchers
Centralized decision making	• Cheaper slave searchers as less memory and processing power needed • Simple to debug	• Failure of a master searcher affects whole region of searchers • Master searcher's performance affects scalability of the number of searchers

Table 2.2: Advantages and disadvantages of centralized/decentralized decision making

	Decentralized decision making	Centralized decision making
Single flat global search map	• Large number of searchers • Hostile environment where UAV failure is possible	• Precision search required
Multiple hierarchical regional search map	• Large search area • Large number of searchers • Hostile environment where UAV failure is possible	• Large search area • Precision search required

Table 2.3: Approach recommendations for scenario requirements

2.3 ARSENL Fixed-Wing UAV (Ritewing Zephyr II)

The platform selected to implement the coordination algorithms is the Ritewing Zephyr II hobby airframe. It is the latest UAV used for ARSENL's swarm UAV field exercises. Figure 2.5 shows a picture of the UAV.

The 56-inch wide UAV has a speed of around 20 meters per second and can fly for approximately 45 minutes. This means the UAV has a flight range of up to 54 km. In the center of the UAV is the processing unit where two processor boards reside. Figure 2.6 reveals

Figure 2.5: ARSENL's fixed-wing swarm UAV (Ritewing Zephyr II)

the processing unit in detail. The two processor boards are based on the ARM architecture described in Section 1.1. The autopilot used is the Pixhawk system from 3DR [9]. The gyroscopes, magnetometers, barometers, accelerometers, and GPS are connected to the Pixhawk where it runs open-source, community-developed software on a real-time operating system (NuttX). The second processor board (autonomy payload) is the ODROID U3 which runs the Robot Operating System (ROS) [28]. ARSENL researchers programmed controllers in Python and run them in the ROS to control UAV behaviors. Section 2.4 describes how various components interact with each other.

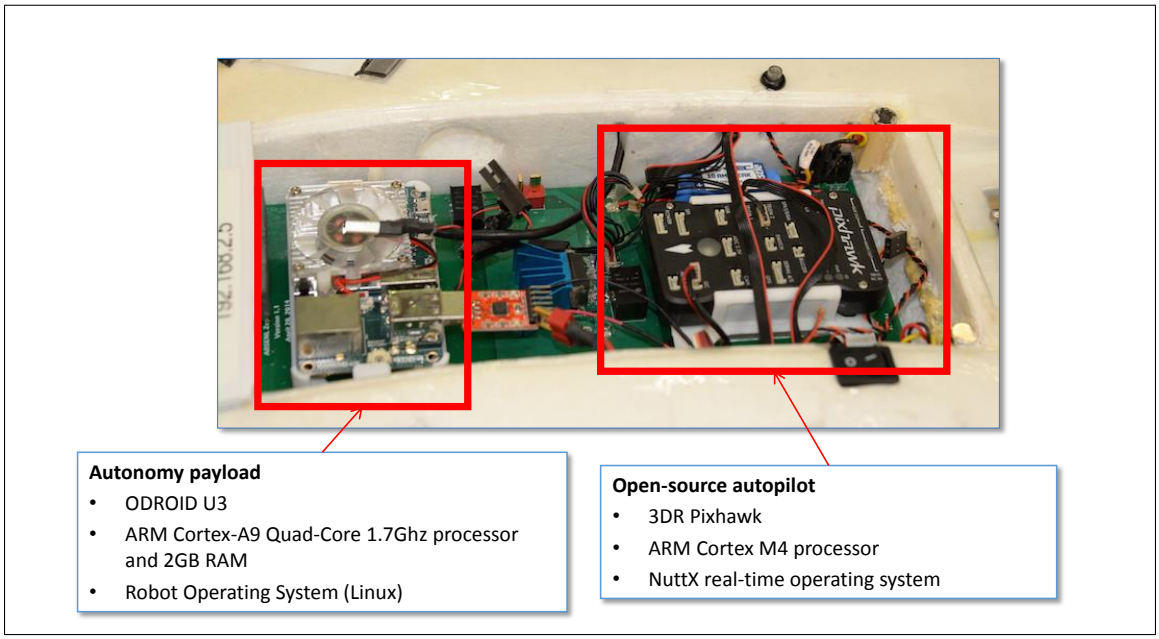

Figure 2.6: Processing unit of the ARSENL's fixed-wing swarm UAV (Ritewing Zephyr II)

2.4 ARSENL Fixed-Wing Swarm UAV System Archiecture

Figure 2.7 shows the system architecture of the UAV. To fly the UAV, the autopilot gathers environment readings from its onboard flight sensors to determine its position, axes, heading, and altitude. It shares this information with the autonomy payload. Depending on which controller is being run in the autonomy payload, the controller decides where the UAV should go next. For the swarming controllers, the autonomy payload can share its positional information with other UAVs in the swarm in network messages via a wireless link (Wi-Fi radio). The autonomy payload may also receive commands from other UAV via the network messages. Once the controller decides where to go, it sends waypoint commands to autopilot where it steers the wings and throttles the motor to fly to the waypoint.

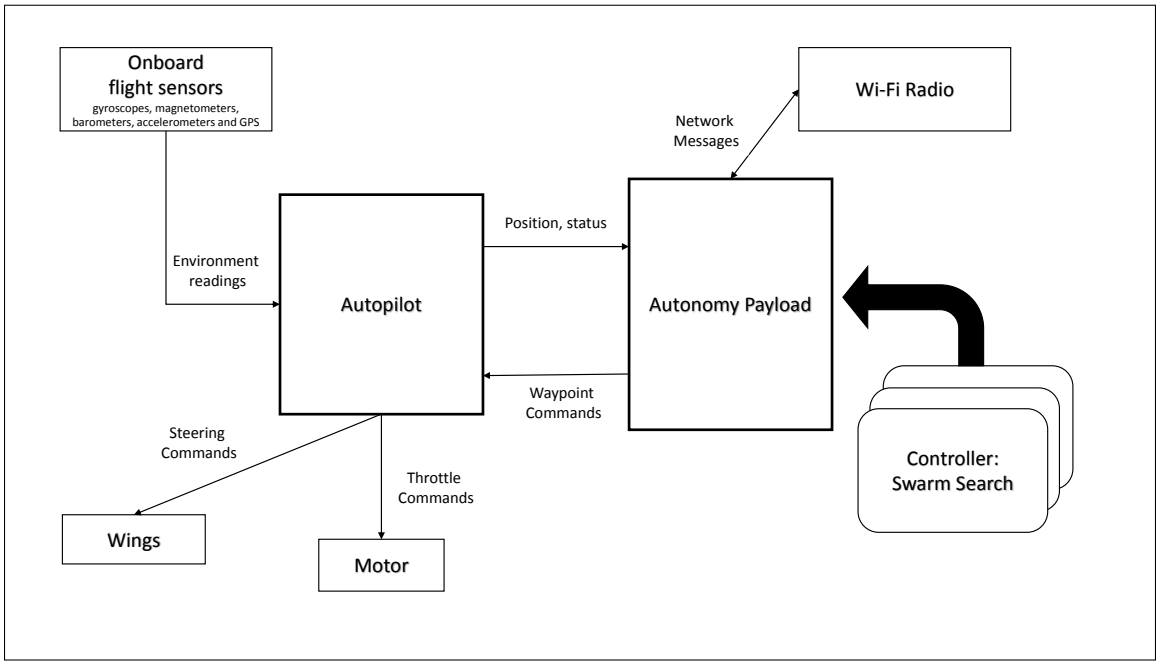

Figure 2.7: ARSENL fixed-wing swarm UAV system architecture

2.5 Swarm Search Controller Design

The bulk of the thesis implementation is in coding the swarm search controller for the autonomy payload. The source code for the swarm search controller is in Appendix: Source Code. A state machine diagram of the swarm search controller is shown in Figure 2.9. When the swarm search controller activates, the UAV searcher will go into the slave searcher state. In this state, the UAV searcher takes on a simple role: its only task is to fly to any waypoint assigned to it. In the meantime, other parts of the autonomy payload broadcast the UAV searcher's positional information on the wireless link. Every UAV searcher can keep track of the position of every other UAV in the swarm.

A swarm search order is issued to the swarm UAV via a network message through the wireless link. This swarm search order contains the information of the search area, which UAV takes the role of the master searcher in the swarm and the swarm search algorithm to use. The user interface to issue this order can be seen in Figure 2.8.

Once the swarm search order is received, the UAV searcher designated to be the master searcher transits to the master searcher state. All other UAV searchers remain as slave

searchers. The master searcher is in charge of the search operation and maintains the global search map. The master searcher starts by generating the search grid based on the search area information in the swarm search order. The Zephyr II UAV has a turning radius of around 70 meters at full speed. So the sensor range of the searchers is assumed to be 75 meters and the search cell's interval is set at 150 meters (2 times the sensor range). Once the search grid is created, the search operation is ready for ingress. All the searchers are then assigned to fly to the center of the search grid. The master searcher issues waypoints to itself also.

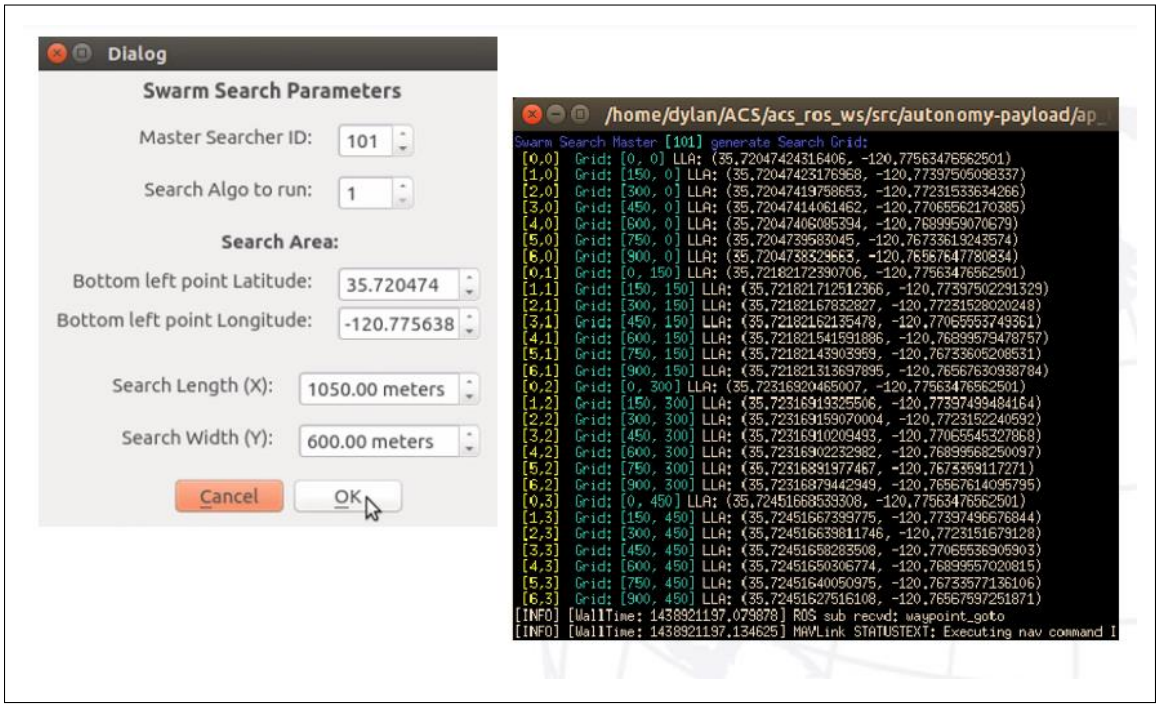

Figure 2.8: User interface to issue swarm search order and the resulting search grid

When one of the searchers arrives within 150 meters of the search grid, the search operation is deemed active and the clock counter for the search starts. This clock counter is used to compare the time for complete coverage of different algorithms. Based on the coverage algorithm used, the master searcher assigns search cells in the search grid to the searchers. Details of the coverage algorithms implemented for this thesis are found in Sections 2.5.1 and 2.5.2. Whenever the master searcher sees that a searcher has visited its assigned search cell, the master searcher updates the global search map and assign other unvisited search

cells to the searcher. When all search cells are visited, the search operation is deemed completed and the clock counter stops. The master searcher then sends all the searchers back to their original positions for the next swarm search order.

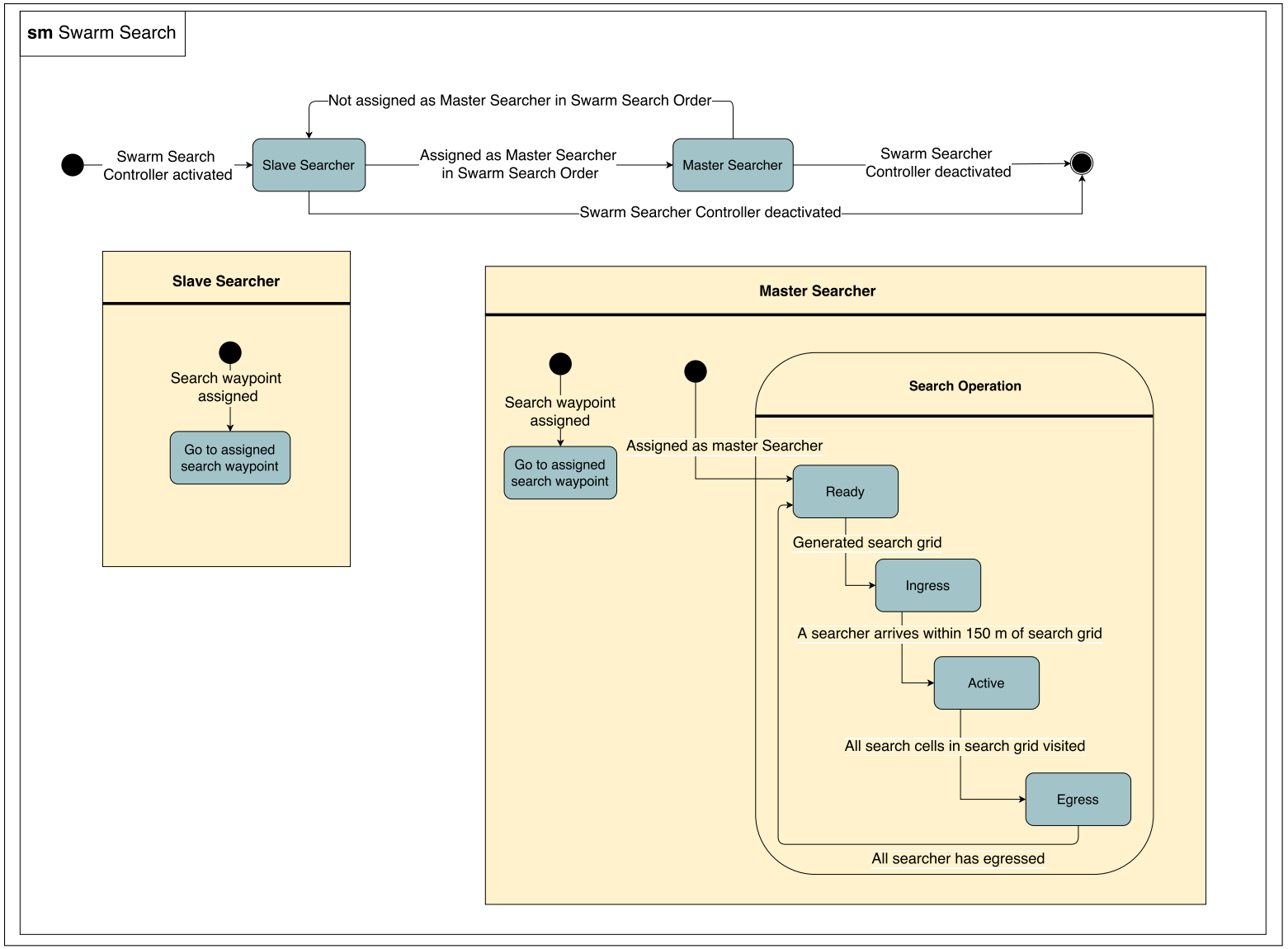

Figure 2.9: State machine diagram of the swarm search controller

2.5.1 Greedy Coverage Algorithm

During the active phase of the search operation shown in Figure 2.9, the master searcher assigns unvisited search cells for the searchers. The way the master searcher assigns the search cells depends on the coverage algorithm specified in the search order. For the purpose of proving the concept of the swarm search controller, a simple "greedy" coverage algorithm is selected for implementation. The greedy coverage algorithm is one of the easiest algorithms to implement and serves well as a benchmark for other future coverage algorithms. The flowchart of this greedy coverage algorithm is shown in Figure 2.10.

When a searcher reaches its assigned search cell, the greedy coverage algorithm just chooses the closest unvisited search cell as the searcher's next search cell. This repeats until all the search cells are visited. The greedy coverage algorithm is used for the validation of the swarm search controller during Field Experiment 1 described in Section 3.5.1.

2.5.2 Fixed Lane Coverage Algorithm

The fixed lane coverage algorithm is designed with the goal of improving upon the greedy coverage algorithm after the successful testing of the swarm search controller in Field Experiment 1. Analysis of the greedy coverage algorithm in Section 4.1 shows that the time for complete coverage can be improved by reducing the flight overlaps between the searchers.

The fixed lane coverage algorithm prevents flight overlaps by pre-allocating search cells to the searchers. The search cells are allocated together as a single lane, either a row or column of the search area. There will be no flight overlaps as each searcher flies a straight line in its own dedicated lane. The limitation of this algorithm is there must be more searchers than the number of lanes from the shortest side of the rectangular search area. If there are more lanes than searchers, the algorithm can not allocate the lanes fully and the entire search has to be conducted with the greedy coverage algorithm instead. Future implementation of the fixed lane coverage algorithm is expected to resolve this limitation.

The flowchart of this fixed lane coverage algorithm is shown in Figure 2.11.

When a searcher reaches its assigned search cell, the fixed lane coverage algorithm chooses the closest unvisited search cell in the searcher's lane as the searcher's next search cell. This

repeats until all search cells in the lane are visited. Since each searcher begins the search at the end of a lane, this approach ensures that each lane is searched without backtracking or overlap. The search operation ends when all the search cells in all searchers' lanes are visited. The fixed lane coverage algorithm is validated during the Field Experiment 2 described in Section 3.5.2.

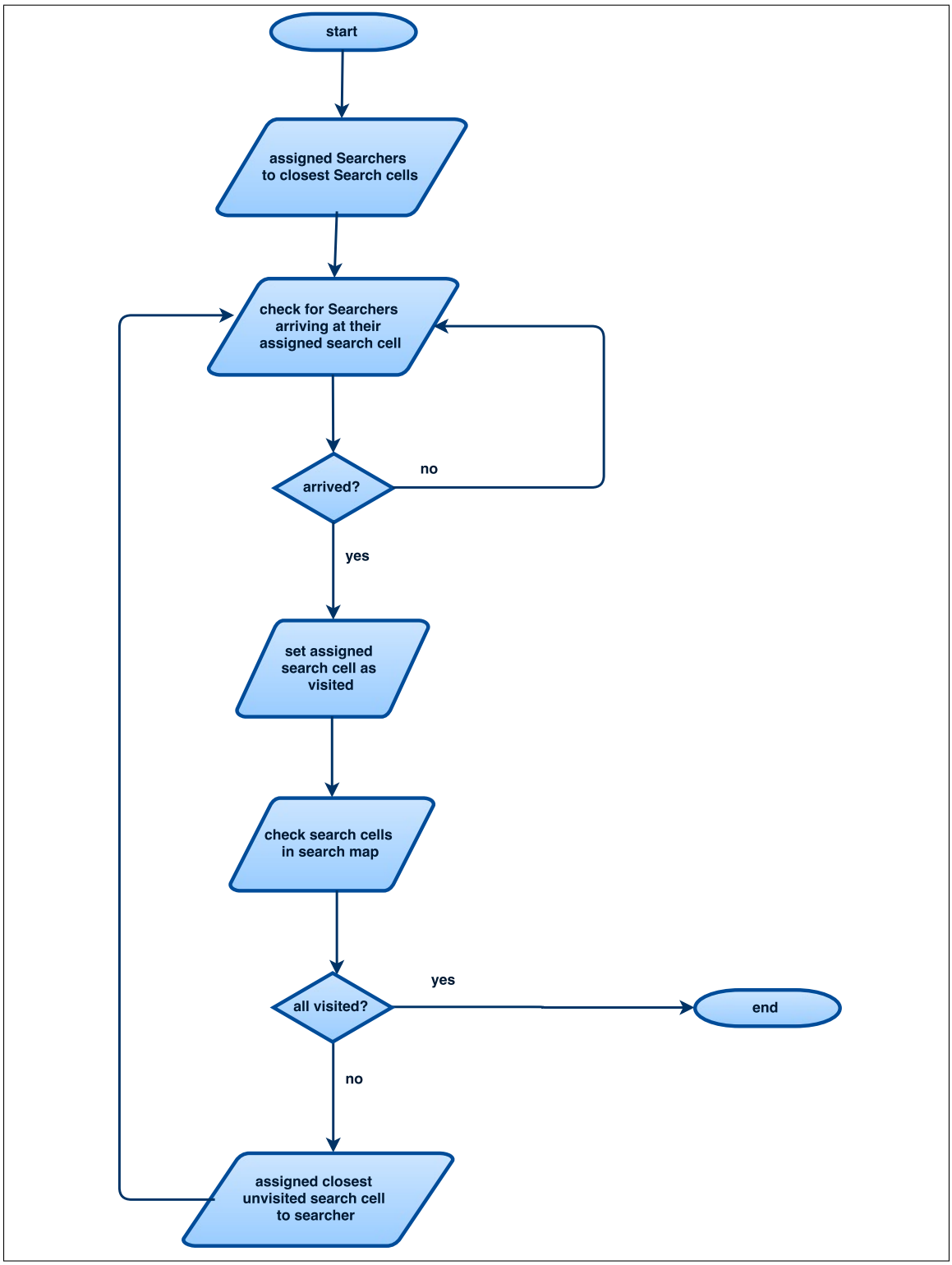

Figure 2.10: Flowchart of greedy coverage algorithm

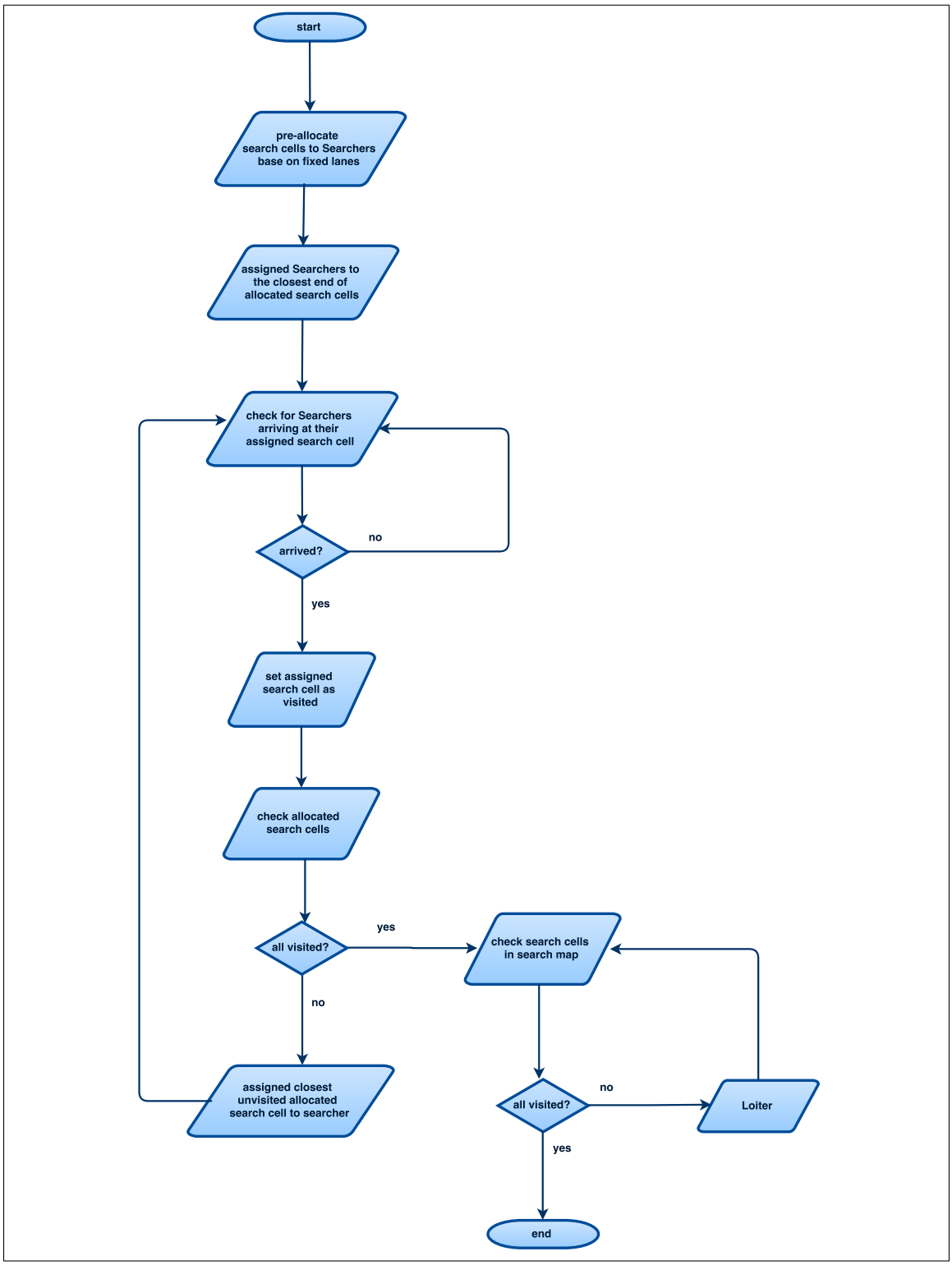

Figure 2.11: Flowchart of fixed lane coverage algorithm

2.6 Coverage Algorithm Evaluation Using Simulation

ARSENL uses JSBSim to simulate the flight of the Zephyr II UAV [29]. JSBSim is an open source flight dynamics model that is integrated with the ARSENL Zephyr II UAV's autopilot software. So instead of receiving and sending commands/information to actual components on the UAV, JSBSim simulates the effect of those commands in the virtual environment and updates the autopilot via simulated readings to the autopilot's sensors. This software-in-the-loop approach allows minimum or no change from development source code to actual deployment source code.

During simulation, visualization of the UAV searchers is provided, as seen in Figure 2.12. The internal state of the swarm search controller can also be printed to the screen to aid debugging and troubleshooting. At the end of each search, the master searcher prints the global search map, as seen in Figure 2.13. The time for complete coverage results and the allocation of search cells among the searcher are used to compare the performance between different coverage algorithms as described in Chapter 4.

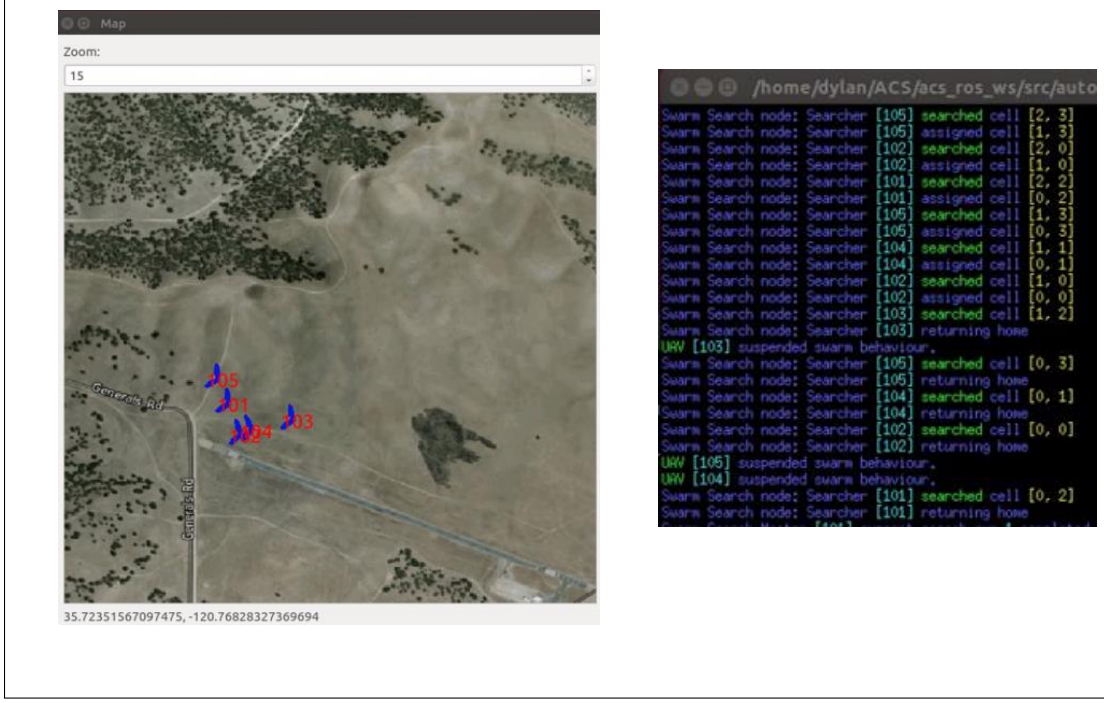

Figure 2.12: ARSENL swarm UAVs simulation environment

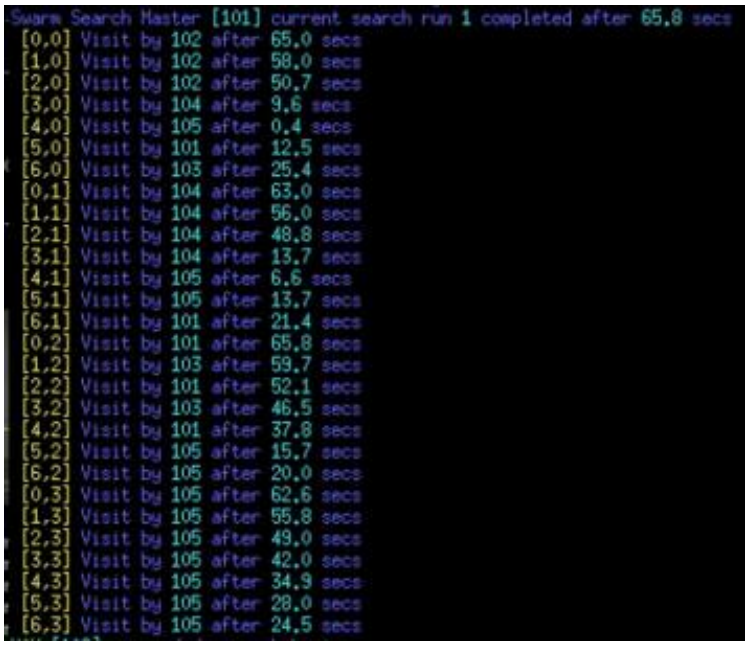

Figure 2.13: Sample coverage algorithm results from simulation

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Experimentation and Results

This chapter presents the scenarios used for the experiment simulation runs and the live-fly field experiments. The results from the simulation runs of the scenarios are shown here. These simulation results are used to evaluate the coverage algorithms in Chapter 4. The events during the live-fly field experiments that are used to validate the swarm search controller and coverage algorithms are described here as well.

3.1 Simulation and Field Scenario

Figure 3.1 shows the scenario, which is the same between the simulation experimentation and Field Experiment 2. The search area is a 900 meter by 600 meter rectangle just north of McMillan Airfield at Camp Roberts, CA. The search area consists of 24 search cells that are 150 meters apart from each other, with the bottom left search cell at latitude 35.720474 and longitude -120.77481. The red boundary shown in Figure 3.1 is the safety fence that encloses the experimentation area. Any UAVs that breach the fence are automatically triggered to return to a rally point near the airfield for remedial action. The swarm UAVs begin each search at the standby point indicated by a white star in Figure 3.1.

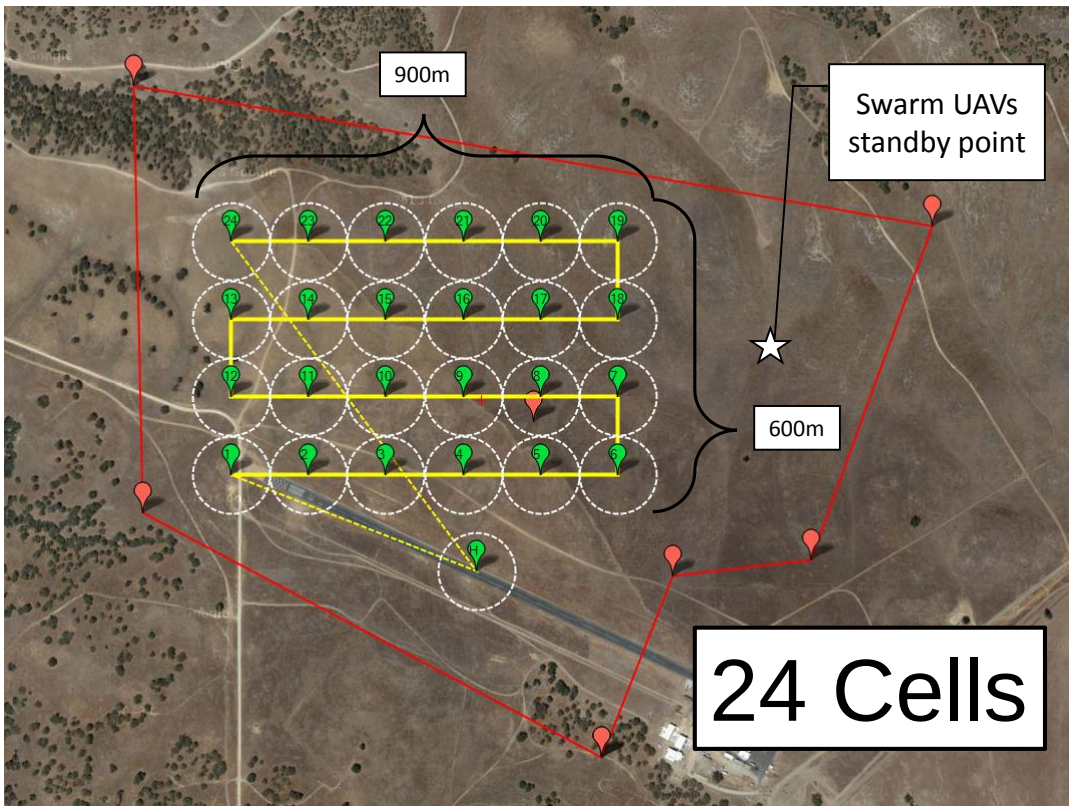

Figure 3.1: Scenario for simulation runs and August 2015 live-fly field experiments

3.2 Simulation Experimentation

The purpose of the simulation experiment is to evaluate and compare the greedy coverage algorithm and the fixed lane coverage algorithm. The effect of the number of searchers on the coverage algorithm is also taken into consideration. The experiment parameters are shown in Table 3.1.

Coverage algorithmn	**No. of searchers**		
Greedy	4	5	6
Fixed lane	4	n.a	6

Table 3.1: Simulation experiment parameters

There are five sets of simulation runs for the experiment parameters: three for the greedy coverage algorithm and two for the fixed lane coverage algorithm. The search area in the

scenario is a six-columns-by-four-rows rectangle. Since the fixed lane coverage algorithm can only allocate lanes according to the number of rows or columns, there is no need to experiment the fixed lane coverage algorithm with five searchers.

It is assumed from the Central Limit Theorem that the time for complete coverage of each set of experiment parameter simulation runs is normally distributed. Having 30 simulation runs for each set of experiment parameters will provide a fairly good approximation of the normal distribution. A good normal distribution is required for assessing the statistical significance between the time for complete coverage means. Thus, a total of 150 simulation runs are made for the experimentation.

3.3 Greedy Coverage Simulation Results

Figure 3.2 shows the scatter chart of the 90 greedy coverage simulation runs.

The mean for the time for complete coverage with four searchers is 56.7 seconds (30 simulation runs). Using 95% confidence intervals, the upper limit of the time for complete coverage with four searchers is 58.5 seconds, while the lower limit is 54.9 seconds. The mean for the time for complete coverage with five searchers is 55.2 seconds. The upper limit of the time for complete coverage with five searchers is 56.8 seconds, while the lower limit is 53.5 seconds. The mean for the time for complete coverage with six searchers is 52.5 seconds. The upper limit of the time for complete coverage with six searchers is 53.8 seconds, while the lower limit is 51.2 seconds.

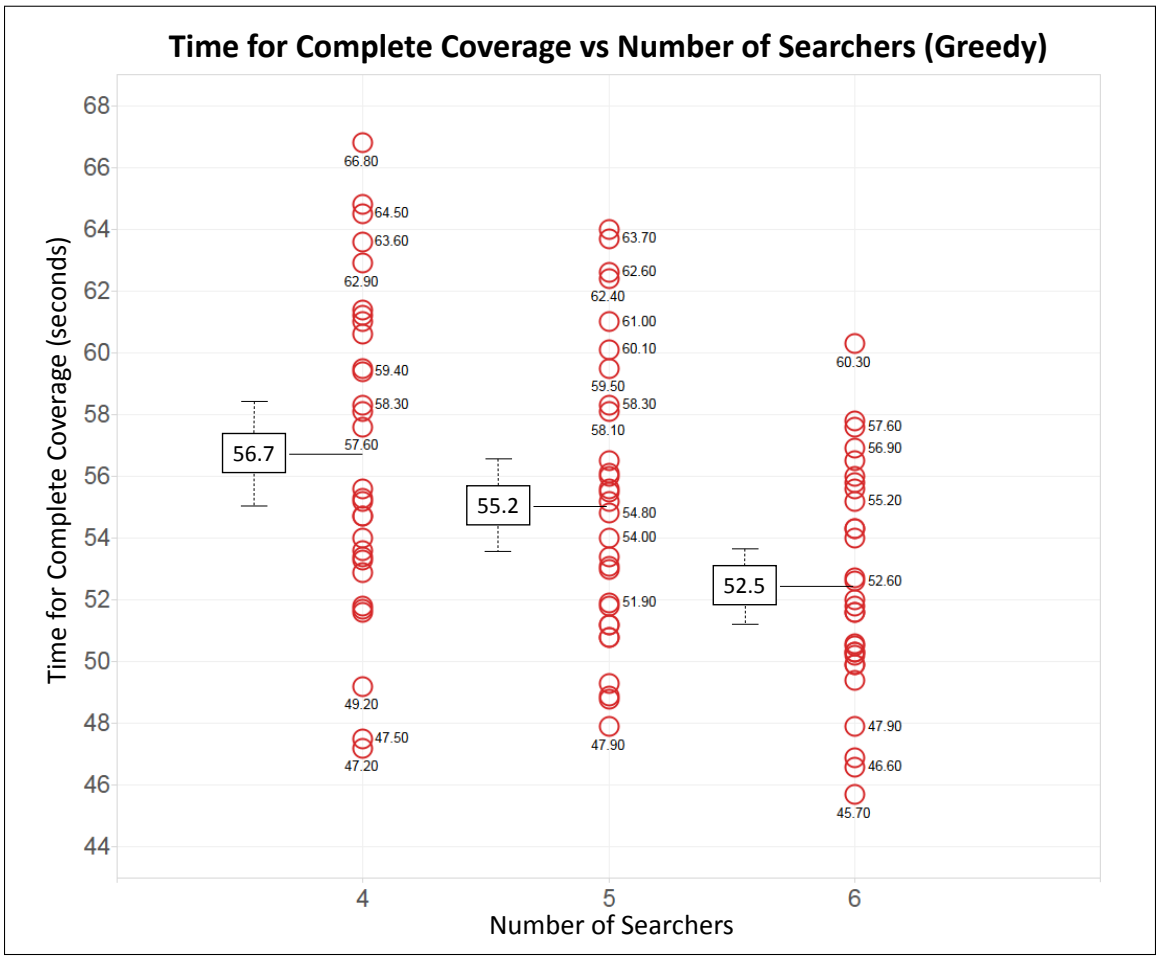

Figure 3.2: Time for complete coverage vs. number of searchers (greedy)

3.4 Fixed Lane Coverage Simulation Results

Figure 3.3 shows the scatter chart of the 60 fixed lane coverage simulation runs.

The mean for the time for complete coverage with four searchers is 49.5 seconds (30 simulation runs). Using 95% confidence intervals, the upper limit of the time for complete coverage with four searchers is 50.1 seconds, while the lower limit is 48.9 seconds. The mean for the time for complete coverage with six searchers is 57.1 seconds. The upper limit of the time for complete coverage with six searchers is 58.4 seconds, while the lower limit is 55.8 seconds.

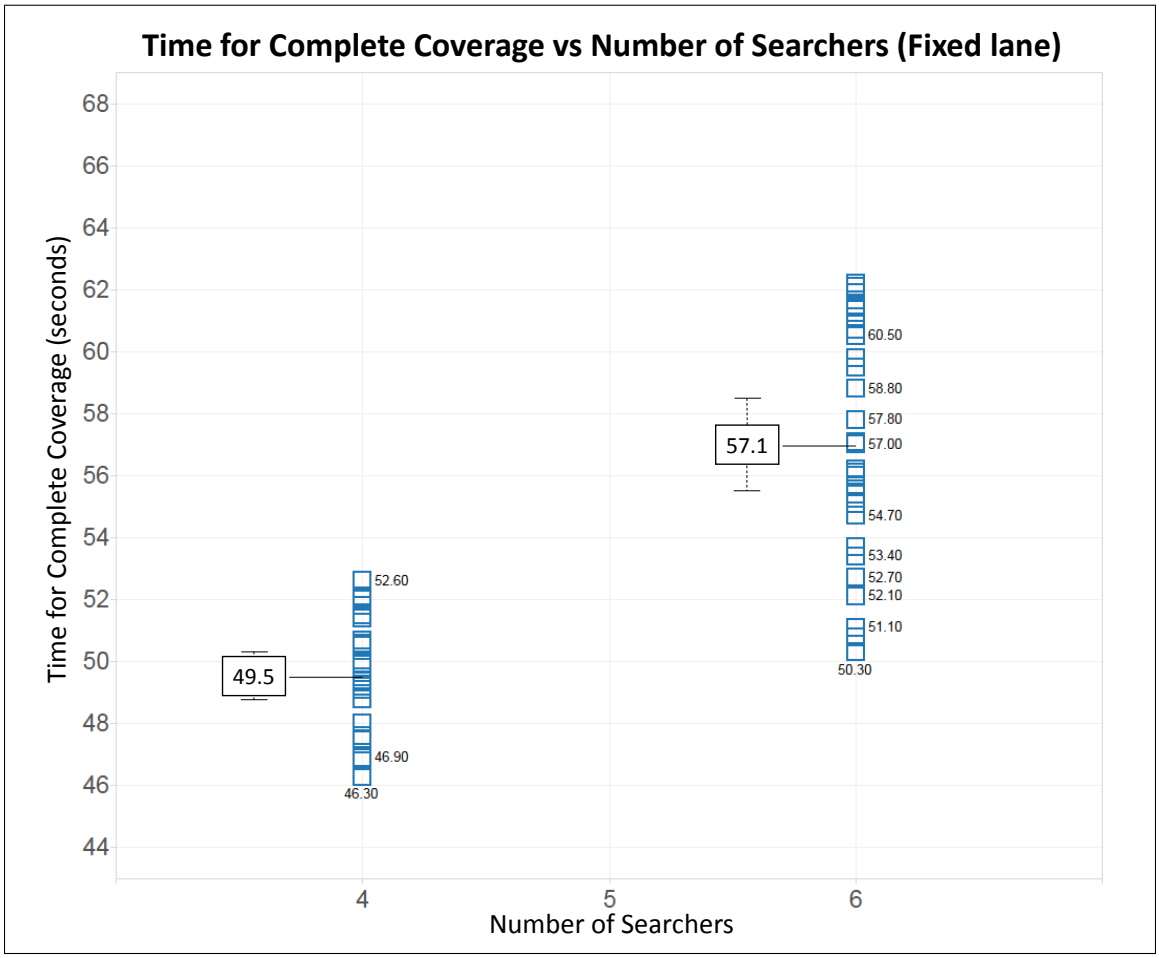

Figure 3.3: Time for complete coverage vs. number of searchers (fixed lane)

3.5 Field Experiment Results

The swarm UAVs field experiments are conducted at McMillan Airfield, Camp Roberts, CA. Camp Roberts' restricted military airspace provides a safe and secure environment for ARSENL researchers to fly large scale swarm UAVs. The McMillan Airfield's 3,500 ft long landing strip is also essential for landing ARSENL's fixed-wing UAVs. Figure 3.4 shows the entrance sign of NPS Field Laboratory at McMillan Airfield.

3.5.1 Field Experiment 1: 14 July 2015

The objective of Field Experiment 1 is to demonstrate this thesis' prototype of an onboard swarm UAV controller to coordinate and conduct an autonomous area search. This is the

Figure 3.4: NPS Field Laboratory entrance sign at McMillian Airfield

first time for ARSENL's swarm UAVs to generate dynamic UAV flight paths using onboard processing to accomplish a mission; the only information required by the swarm UAVs are the size and location of the search area. Even though the controller is tested successfully in simulation, there is always a risk of real-world issues not captured in the simulation:

- whether computation by the controller is sustainable and scalable
- whether actual communications conditions (e.g., latency or losses) will support the coordination between the UAVs
- whether the flight systems will be able to accomplish the commands given to them by the controller

The search area in the Field Experiment 1 scenario is a 600 meter by 450 meter rectangle with 12 search cells, as shown in Figure 3.5.

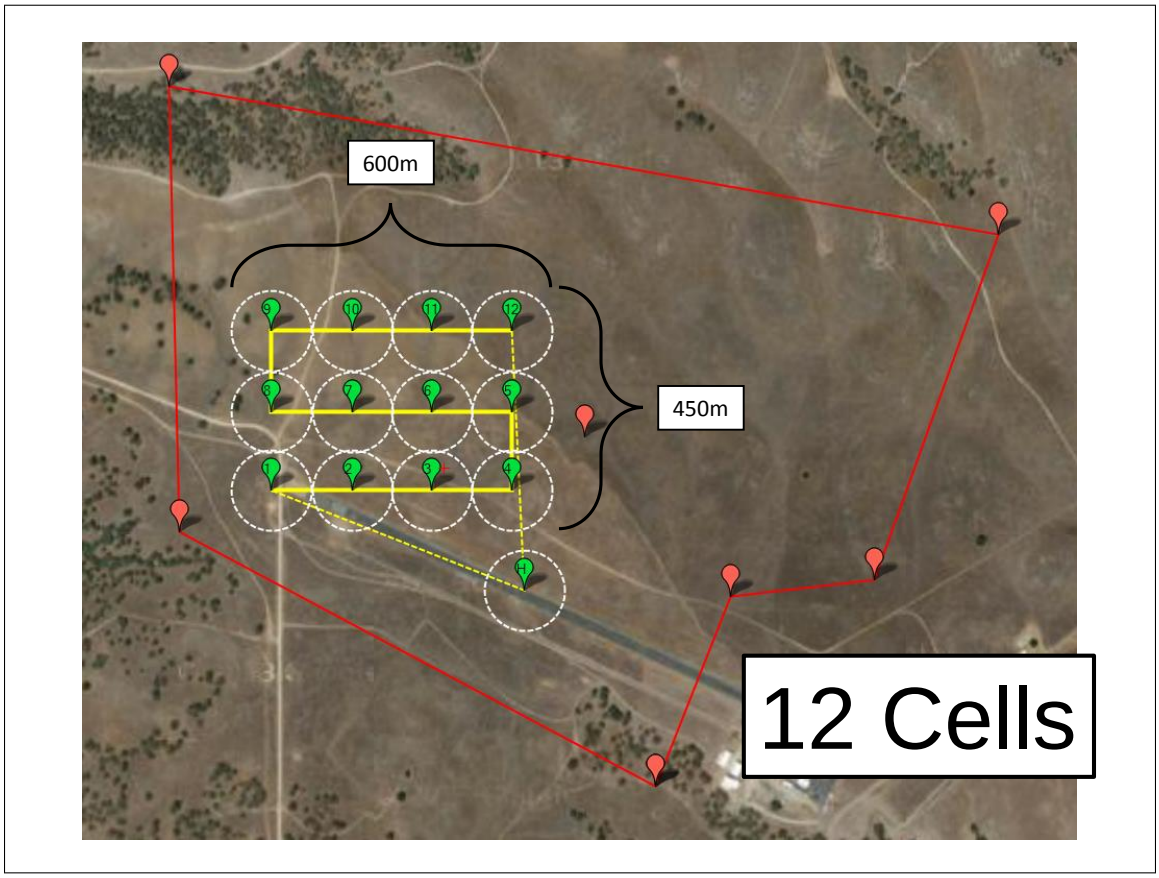

Figure 3.5: Scenario used for the 14 July 2015 field experiments

Five UAVs were used as the swarm UAVs in the field experiment. Figure 3.6 shows the five UAVs being ready for the swarm search controller's flight trials.

One of the UAVs failed to take off from the launcher, but the concept of the swarm search controller coordinating the swarm UAVs for area search was still proven successfully with the four remaining UAVs. The master searcher flying in the swarm UAVs generated the search grid and assigned search cells to the slave searchers using the onboard Wi-Fi radios. Figure 3.7 shows the full flight trajectories taken by the four UAVs during Field Experiment 1.

Figure 3.6: Five UAVs used to validate the swarm search controller running greedy coverage algorithm

3.5.2 Field Experiment 2: 26 August 2015

The first objective of Field Experiment 2 is to test the swarm search controller with a larger search area. The search area is doubled from 12 search cells from Field Experiment 1 (Figure 3.5) to 24 search cells (Figure 3.1). The second objective of Field Experiment 2 is to validate the fixed lane coverage algorithm. Five UAVs were used as the swarm UAVs in the field experiment. The swarm UAVs completed the area search successfully using the fixed lane coverage algorithm and the greedy coverage algorithm. Both objectives of Field Experiment 2 were met. Figure 3.8 shows the full flight trajectories taken by the five UAVs during Field Experiment 2.

3.5.3 Future Field Experiment

A test plan is created for future field experiment to validate the simulation results findings. The test plan is shown in Table 3.2.

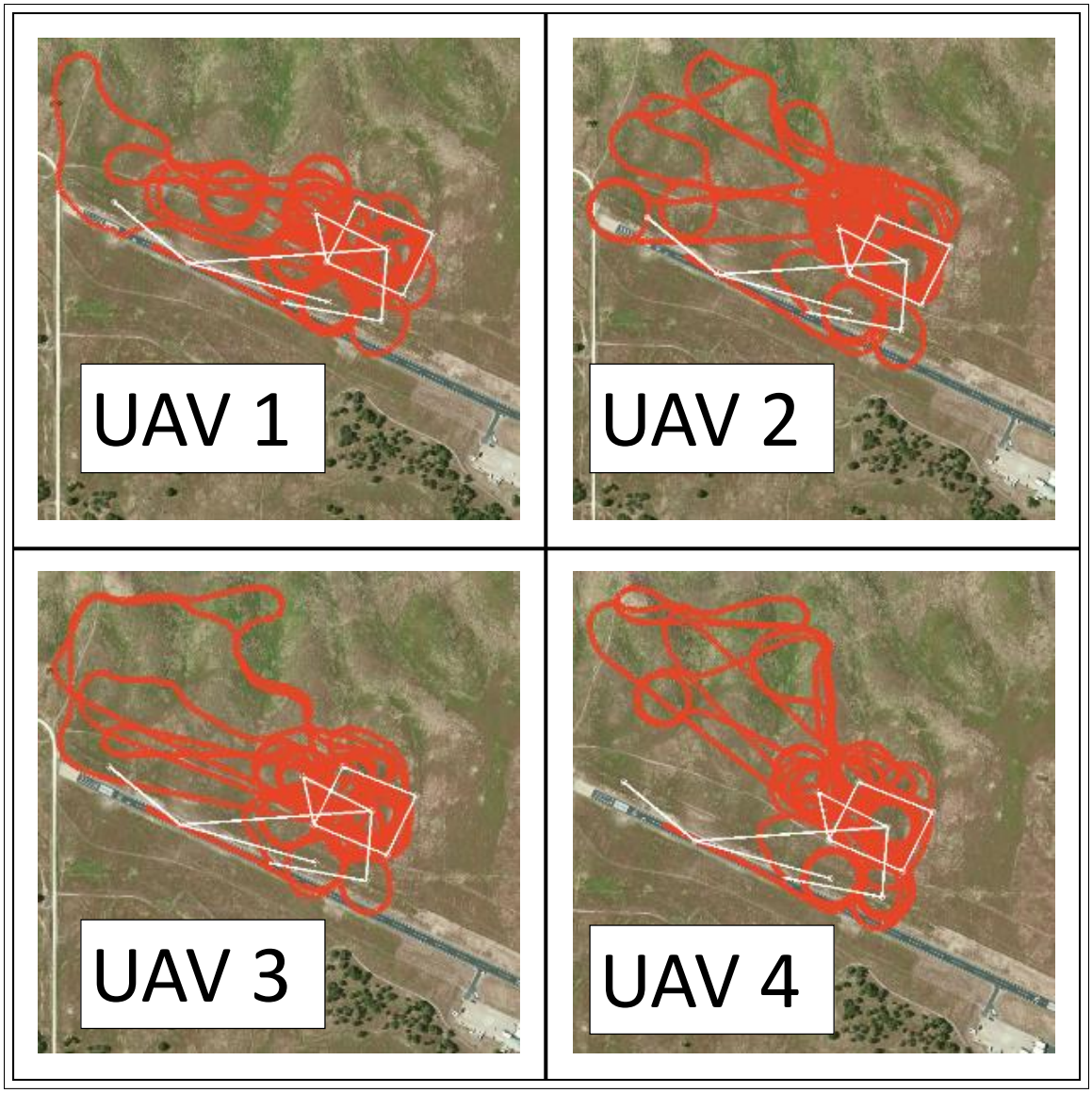

Figure 3.7: Full flight trajectories taken by the swarm search UAVs during Field Experiment 1

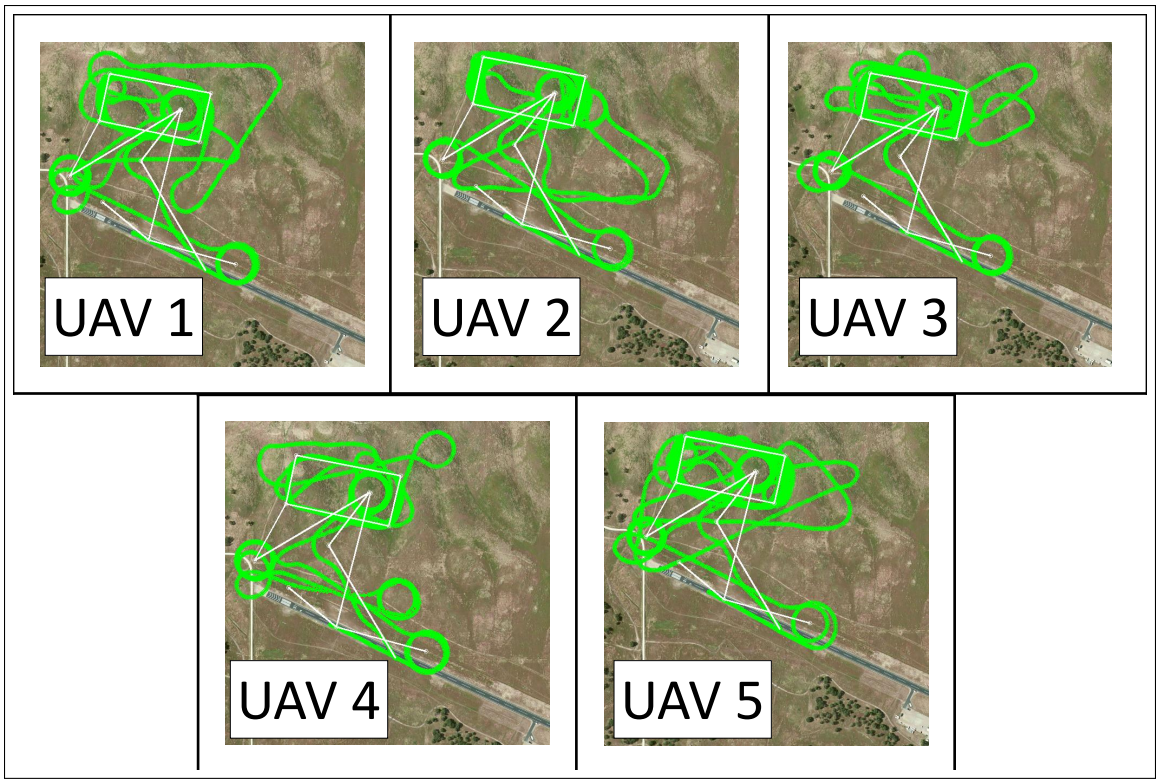

Figure 3.8: Full flight trajectories taken by the swarm search UAVs during Field Experiment 2

Test name	Purpose	Expected Results	Live-fly Results
Greedy with 6 UAVs	Gather coverage time for greedy with 6 UAVs	Search completed with all UAVs returning to swarm standby point	Pass?
Fixed lane with 6 UAVs	Gather coverage time for fixed lane with 6 UAVs	Search completed (columns) with all UAVs returning to swarm standby point	Pass?
Fixed lane with 5 UAVs	Test the swarm search controller in retiring the extra UAV; gather coverage time for fixed lane with 4 UAVs	Search completed (rows) with 4 UAVs returning to swarm standby point	Pass?
Greedy with 5 UAVs	Gather coverage time for greedy with 5 UAVs	Search completed with all UAVs returning to swarm standby point	Pass?
Greedy with 4 UAVs and greedy with 2 UAVs	Test the swarm search controller in coordinating multiple area search at the same time; gather coverage time for greedy with 4 UAVs	Search completed with 4 UAVs returning to swarm standby point while the other 2 UAVs are still searching	Pass?
Greedy with 2 UAVs (from previous run) and fixed lane with 4 UAVs	Test the swarm search controller in coordinating multiple area search with different coverage algorithms at the same time	Search completed with all UAVs returning to swarm standby point	Pass?
Greedy with 10-15 UAVs	Test the swarm search controller's ability to handle 10-15 UAVs	Search completed with all UAVs returning to swarm standby point	Pass?

Table 3.2: Future field experiment test plan

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Simulation Result Analysis

This chapter describes the analysis on the simulation experimentation results. The purpose of the experiment is to evaluate and compare the greedy and fixed lane coverage algorithms in the swarm search controller for an area search. Of the criteria to evaluate coverage algorithms listed in Section 1.4, "time until complete coverage," "missed coverage," and "load balancing" are examined in this thesis. The other criteria, "communication requirements," "communication robustness," and "computational requirements" are not as critical considering that both coverage algorithms are demonstrated successfully in live-fly field experiments. The scenario used for the simulation is the same as the Field Experiment 2.

4.1 Time for Complete Coverage Analysis

A total of 150 simulation runs are used for the analysis. There are 30 simulation runs for each set of the five experiment parameters. The times for complete coverage for all the simulations runs are consolidated in a scatter chart shown in Figure 4.1. The mean for each set of experiment parameters is indicated on the chart along with the 95% confidence intervals upper and lower limits.

4.1.1 Having More Searchers Does Not Guarantee Shorter Time for Complete Coverage

Figure 4.1 seems to indicate that having more searchers when using the greedy coverage algorithm reduces the time for complete coverage. However, the only statistically significant results to support this hypothesis are when six searchers using the greedy coverage algorithm are compared to four searchers using the same greedy coverage algorithm.

When using the fixed lane coverage algorithm, the opposite was actually true. The four searchers using the fixed lane coverage algorithm completed the coverage in significantly less time than the six searchers using the fixed lane coverage algorithm. This is a counter-intuitive finding. The reason why the six searchers took more time than four searchers was because of the searchers' starting position (standby point). The starting position was

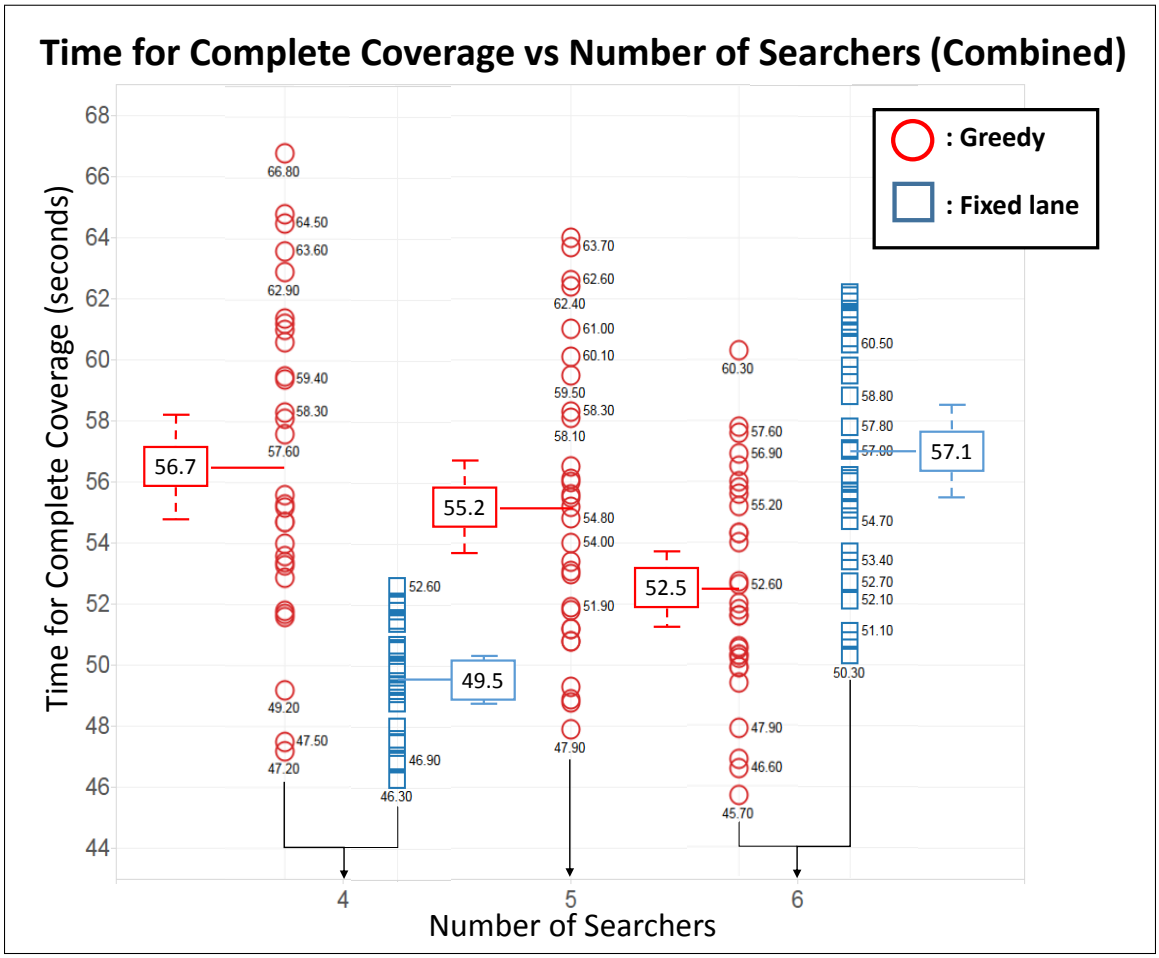

Figure 4.1: Time for complete coverage vs. number of searchers (combined)

east of the search area. This placed the searchers along the row's axis and allowed the four searchers to "slide" into the search area immediately, as shown in Figure 4.2. On the other hand, when using six searchers, the searchers needed to pre-position at one end of the column before they could enter the search grid; this means one of the six searchers always needed to fly across the search area to the last column on the other side before flying down or up the column. This is shown as UAV "1" in Figure 4.3.

It should also be noted that were the search area re-oriented (i.e., six rows versus four columns), the six searchers' time for complete coverage would have outperformed the four searchers' time for complete coverage by a wide margin because the six searchers would have less transit time and covered the search area more efficiently.

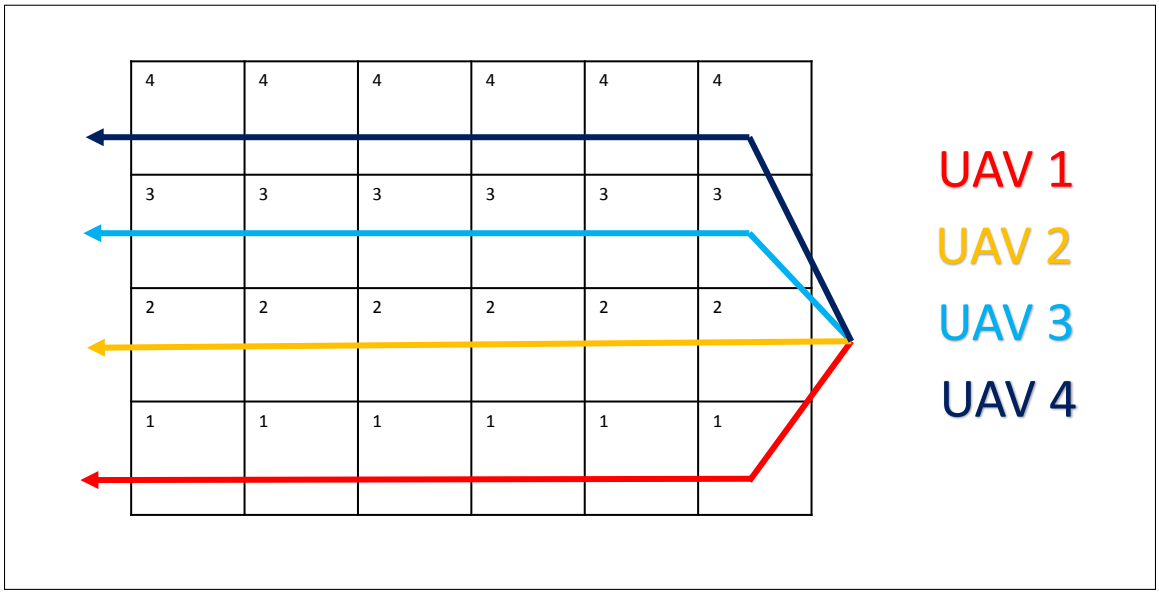

Figure 4.2: Flight paths of fixed lane coverage algorithm with four searchers

This finding suggests that the relative position of the searchers' starting position to the search area can have a significant impact to the time for complete coverage. A potential improvement to the fixed lane coverage algorithm is the search master determining whether the search area will better searched by row or column. The search master considers the number of searchers and the orientation of the search area before choosing to search by row or column.

4.1.2 Consistent Performance

The standard deviations for the greedy coverage algorithm with four, five, and six searchers are 5.11, 4.58, and 3.56 respectively. The standard deviation for the fixed lane algorithm with four and six searchers are 1.65 and 3.62, respectively. A smaller standard deviations means that there is less variance in the results of that experiment parameter. This is shown in Figure 4.1.

The greedy coverage algorithm gives more consistent results when more searchers are involved in the search. For the fixed lane coverage algorithm, having more searchers gives less consistent results. The reason for this difference is because the six searchers using fixed lane coverage algorithm have a longer transit time than the four searchers using the fixed

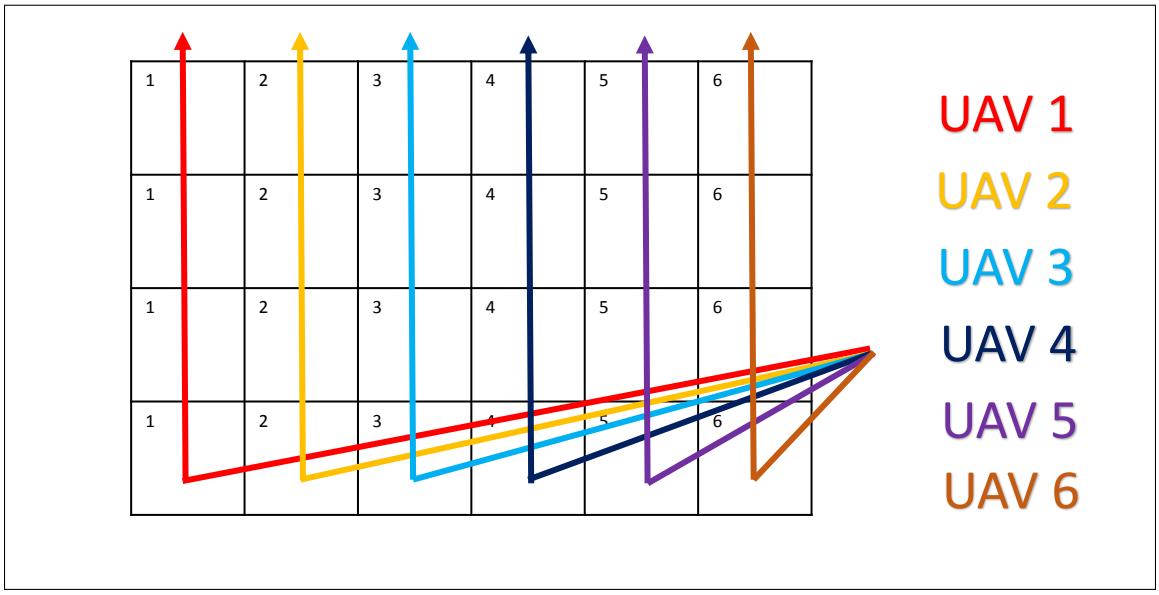

Figure 4.3: Flight paths of fixed lane coverage algorithm with six searchers

lane coverage algorithm. The additional variance in the searchers' transit time reduces the consistency of the time for complete coverage.

4.1.3 Shortest Time for Complete Coverage

From Figure 4.1, the shortest time for complete coverage is the fixed lane coverage algorithm with four searchers. The fixed lane coverage algorithm with six searchers fared poorly because of their starting position. The greedy coverage algorithms had a longer time for complete coverage because of the overlaps. This is shown in Figure 4.4. Based on the equation derived in Section 1.3,

$$T = \frac{A}{VWN}$$

where the "perfect" area search time for the scenario with four searchers is as follows:

$$T = \frac{900\ m \times 600\ m}{20\ m/s \times 150\ m \times 4} = 45\ seconds \tag{4.1}$$

The mean of the fixed lane coverage algorithm with four searchers is 49.5 seconds, which

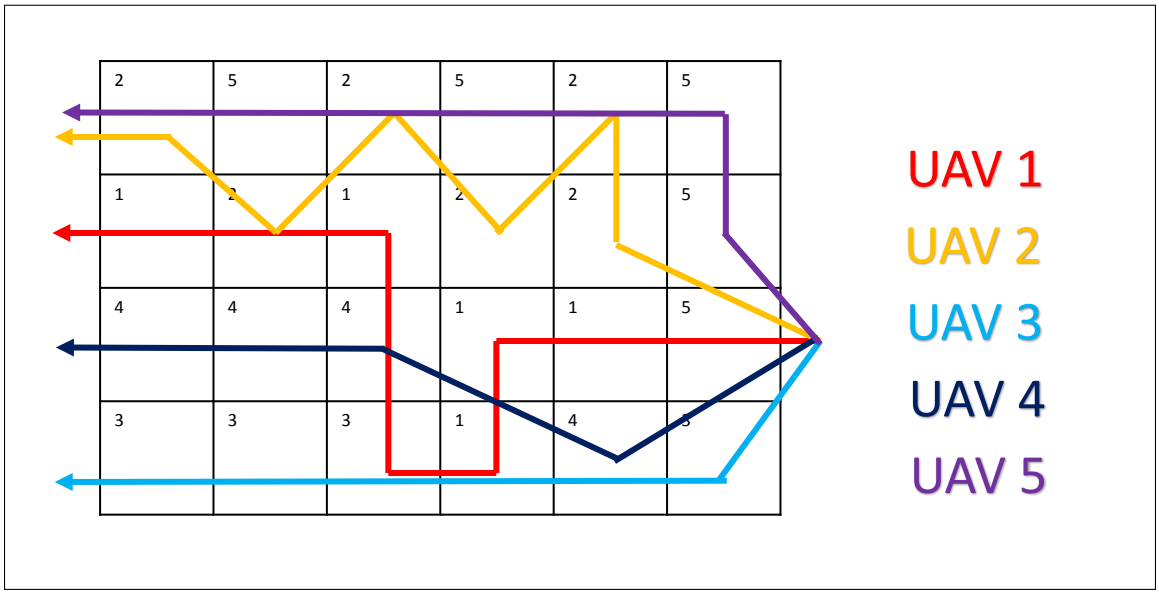

Figure 4.4: Sample flight paths of greedy lane coverage algorithm with five searchers

is close to the perfect area search time. It is still possible for the greedy coverage algorithm to have a shorter time for complete coverage though. There are 24 search cells in the search area. If the greedy coverage algorithm has 24 searchers, all the searchers only have to fly directly to one search cell and they will achieve a shorter time than the fixed lane coverage algorithm. There will be significant overlap in coverage though.

4.2 Missed Coverage and Overlap Analysis

The fixed lane coverage algorithm prevents any overlapping between the searchers by allocating dedicated lanes to the searchers. The sweep width is the same as the search cells interval so there are no overlaps and gaps. The gap that appears in the corner when a searcher turns (Figure 2.3) is also prevented as the fixed lane coverage algorithm does not require any of the searchers to make any turns. The greedy coverage algorithm cannot prevent overlap and missing coverage due to turns by the searchers.

4.3 Load Balancing Analysis

The fixed lane coverage algorithm divides the search cells equally among the searchers so the load is balanced evenly among the searchers. The greedy coverage algorithm, on the

other hand, does not consider load balancing at all. The results from the 90 greedy coverage algorithm simulation runs, however, indicates that the load between the searchers is shared more evenly when the number of searchers increases. This result is shown in Figure 4.5.

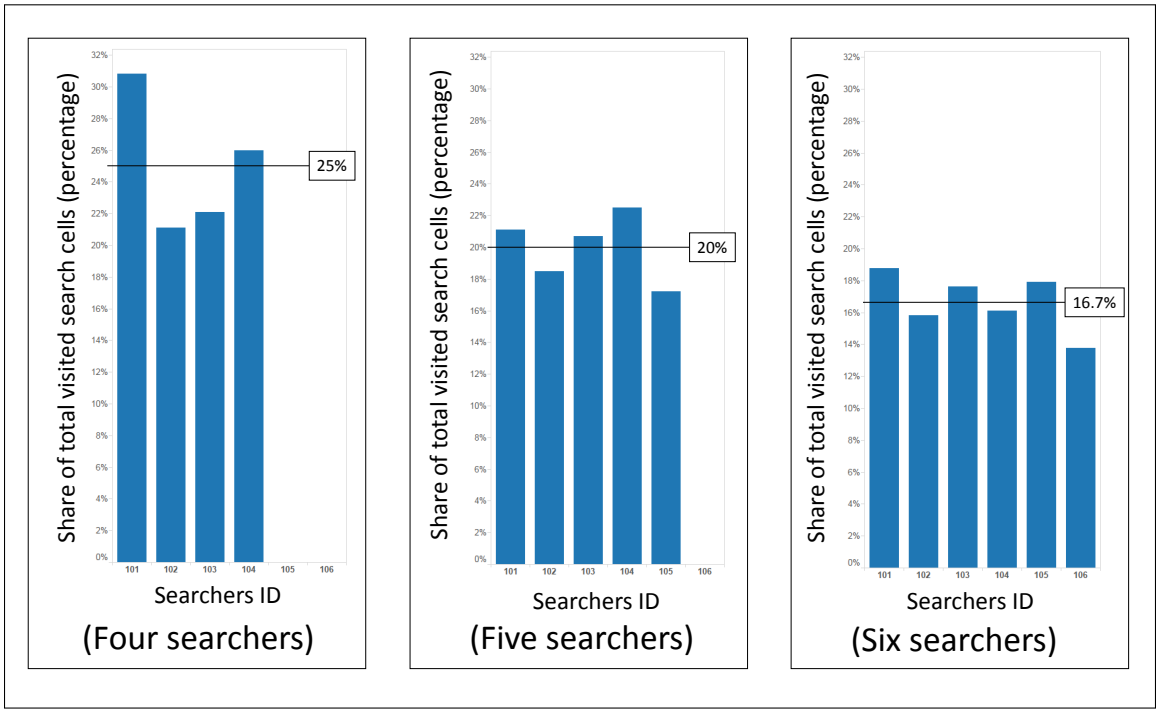

Figure 4.5: Comparison of greedy coverage algorithm load balancing by number of searchers

CHAPTER 5:
Conclusion

This thesis explains the rise of swarm UAVs and the motivation for using swarm UAVs to conduct an area search. The thesis identified the key challenges of coordinating swarm UAVs for an area search. The thesis examined possible approaches to overcoming these challenges, and the most suitable approach for the thesis was selected for implementation. A new swarm search controller, programmed to run onboard swarm UAVs, validated the selected approach with live-fly field experiments. Different coverage algorithms, programmed into the swarm search controller, were evaluated using simulation. Subsequent live-fly field experiments validated the coverage algorithms.

5.1 Summary

The technological advances brought by the smartphone revolution paved the way for the proliferation of hobbyist UAVs. The accessibility of hobbyist UAVs in recent years brings the opportunity for researchers to deploy swarm UAVs. Swarm UAVs have the potential to improve current surveillance or search missions by collecting data from multiple vantage points simultaneously. Researchers have been steadily increasing the swarm size and complexity of the swarm UAVs they are studying. However, autonomous outdoor swarm behaviors are still restricted to formation flying or "flocking." This thesis seeks to advance the field by coordinating autonomous outdoor swarm UAVs for an area search.

Theoretical work identifies that the key challenges to coordinating swarm UAVs for area searches are, preventing the swarm UAVs from overlapping with each other during the search and ensuring that all of the search area is covered. Several approaches were discussed to overcome the two challenges. The following approaches were found to be most suitable for the thesis to implement. To prevent overlaps, a global search map of all swarm UAV positions was maintained through sharing of positional information using a wireless link. A centralized master searcher used the global search map to decide the search path for itself and other swarm UAVs. The coverage of the search area was tracked by discretizing the search area.

For this thesis, a new swarm search controller was designed and programmed to coordinate swarm UAVs for an area search. The swarm search controller runs onboard the ARSENL fixed-wing swarm UAV (Ritewing Zephyr II). The swarm search controller's role in the ARSENL UAV System Architecture is explained, including details about the controller's software design. The swarm search controller coordinated area search missions successfully during live-fly field experiments and validated the thesis' approaches.

Two coverage algorithms were implemented in the swarm search controller and validated during the live-fly experiments. The first coverage algorithm is a simple greedy algorithm that assigns discretized search cells to swarm UAVs based on closest distance. The second coverage algorithm were a fixed lane algorithm that minimizes overlaps by pre-allocating discretized search cells to swarm UAVs. For this study, 150 simulation runs were made using both coverage algorithms with different numbers of searchers. The results from the simulation runs were analyzed for statistical significance, and future live-fly experiments are planned to validate the simulation findings.

5.2 Main Findings

This thesis' main finding is that it is possible for outdoor swarm UAVs to coordinate and generate dynamic flight paths using onboard processing to complete an area search mission. Theoretical work shows the key problems to solve for coordinating swarm UAVs to conduct an area search. The problems are preventing the swarm UAVs from overlapping with each other during the search and ensuring coverage of all of the search area.

Analysis of the simulation results reveals that having more searchers does not guarantee a shorter time for complete coverage. The relative position of the searchers' starting position to the search area and the orientation of the search area can have a significant impact to the time for complete coverage. The greedy coverage algorithm had more consistent results when more searchers were involved in the search, while having fewer searchers for the fixed lane coverage algorithm gave more consistent results. Overall, the fixed lane coverage algorithm usually requires a shorter time for complete coverage than the greedy coverage algorithm.

5.3 Future Work and Recommendation

A new paradigm is opened up by this thesis as it shows that swarm UAVs are ready for software experimentation. The thesis demonstrated this readiness by experimenting with different coverage algorithms in the swarm search controller. Further work to improve the software algorithms in the swarm search controller is recommended.

The fixed lane coverage algorithm produces a time for complete coverage that is very close to the perfect time for complete coverage, calculated using the equation explained in Section 1.3:

$$T = \frac{A}{VWN}$$

However, the fixed lane coverage algorithm requires the number of searchers to be more than the number of lanes from the shortest side of the rectangular search area. A potential future work is to find another coverage algorithm that produces the same or better time for complete coverage and does not have the limitation on the size of the search area.

The swarm search controller uses a centralized master searcher decision-making approach. If the size of the swarm UAVs increases a lot, a decentralized decision-making approach may be necessary to prevent the swarm UAVs from having a bottleneck at the master searcher. A potential future work is to convert the swarm search controller to be decentralized.

It is foreseeable that the swarm UAVs will be used to search for targets rather than just area searches. However, real-time video surveillance is a complex task and requires heavy processing. The ODROID U3 autonomy payload onboard the ARSENL swarm UAVs may not have enough processing power to handle the task. A "smart" camera can be utilized to handle the task. The smart camera captures image and also processes them [30]. A potential future work is to integrate the smart camera with the ODROID U3 and have the swarm UAVs search for a target.

The thesis findings are based on the 900 meter by 600 meter search area. The search area used is small enough to fit within Wi-Fi radio range of any swarm UAVs. If the search area expands beyond Wi-Fi radio range, the master searcher may not receive the positional information from the slave searchers to form the correct global search map. Likewise,

the slave searchers may not receive the search cell assignment from the master searcher. A potential future work is to implement the hierarchical regional search maps, which are shown in Table 2.1, to extend beyond Wi-Fi radio range.

5.4 Conclusion

The thesis has shown that research in the swarm UAV field is no longer about realizing the swarm UAVs. The field has matured enough that researchers can shift their focus to pursuing actual real-world applications for swarm UAVs. The live-fly validated swarm search controller represents the first steps in advancing this new focus. By experimenting with different coverage algorithms in the swarm search controller, the thesis opens up a new paradigm by proving that swarm UAVs are ready for software experimentation. This research will undoubtedly be joined by many new and exciting contributions from future researchers.

APPENDIX: Source Code

The soft copy of this code can be accessed at: http://faculty.nps.edu/thchung (under Resources)

sourceCodes/Swarm_Searcher_ctrl.py

```
1  # Standard python library imports
2  import sys
3  import time
4  import math
5  from argparse import ArgumentParser
6
7  # ROS library imports
8  import rospy
9  import std_msgs.msg as stdmsg
10
11 # ACS imports
12 import ap_msgs.msg as apmsg
13 import ap_srvs.srv as apsrv
14 import ap_lib.ap_enumerations as enums
15 import ap_lib.nodeable as nodeable
16 import ap_lib.waypoint_controller as wp_controller
17 import autopilot_bridge.srv as apbrgsrv
18 import autopilot_bridge.msg as apbrgmsg
19 from ap_lib.gps_utils import *
20
21 # Base name for node topics and services
22 NODENAME = 'swarm_searcher'
23
```

```
24 # Global variables (constants)
25 # Local Enumeration for swarming uav search states
26 SEARCH_READY = 1    # Awaiting search area
27 SEARCH_INGRESS = 2 # Received search area and transiting to it
28 SEARCH_ACTIVE = 3  # Searching in search area
29 SEARCH_EGRESS = 4  # search end/completed and transiting to staging area
30 SEARCH_TRACKING = 5# search uav detected something is tasked to track it
31 SEARCH_FAULT = 99  # search uav has shown a fault and kept out of the operation
32
33 SEARCH_CELL_FULLY_VISITED = 0 # All cells visited
34 SEARCH_CELL_FULLY_ASSIGNED = 1 # All cells assigned
35 SEARCH_CELL_STILL_AVAILABLE = 2 # Cells still available for assignment
36
37 SEARCH_CELL_RADIUS = 150 # Interval between search cell in metres
38 SEARCH_ALTITUDE = 50 # Height to conduct search in metres
39 SEARCH_SAFETY_ALTITUDE_INTERVAL = 15 # Height between UAV of the same cell in metres
40 SEARCH_CELL_THRESHOLD = 100 # Distance to determine that cell is searched (must be greater than
       infinite loiter radius
41 SEARCH_ARRIVAL_THRESHOLD = 150 # Distance to start assigning cells in search grid to UAV
42 SEARCH_MAX_THRESHOLD = 100000000 # Max number for comparsion
43 SEARCH_TIMEOUT_INTERVAL = 15 # 15 secs interval to check if searcher get closer to assigned
       waypoint
44 SEARCH_TIMEOUT_STRIKES = 3 # Number of strikes a UAV can get from timeout before being thrown out
        of the search operation
45 SEARCH_TIMEOUT_LOITERDISTANCEBUFFER = 10 # Buffer for UAV doing loiter such as Egressing after
       reaching the swarming waypoint
46
47 SEARCH_SWARM_SEARCH_READY_STATE = enums.SWARM_READY # enum state when UAV is ready for swarm
       behavior
```

```
48 SEARCH_SWARM_SEARCH_ACTIVE_STATE = enums.SWARM_ACTIVE # enum state when UAV is executing an
       active swarm behavior
49
50 # "struct" to store data for search search cell
51 class cell():
52     LLA=None
53     grid=None
54     assigned=False
55     assignedTo=None
56     visited=False
57     visitedBy=None
58     visitedTimeStamp=None
59
60 # "struct" for Waypoint Msg
61 class wpMsg():
62     recipientvehicle_id = None
63     waypoint = apbrgmsg.LLA()
64     searchCell_x = None
65     searchCell_y = None
66
67 # "struct" to info of each search UAV
68 class searchUAVdata():
69     status=None
70     subSwarmID=None
71     receivedState=None
72     pose=None
73     assignedAltitude=None
74     assignedCell=None
75     assignedTime=None
```

```
76      assignedLLA=None
77      preplannedPath=0
78      originalLLA=None
79      timeoutdistance=None
80      timeoutTime=None
81      timeoutCounter=0
82
83  # Class member functions:
84  class SwarmSearcher(wp_controller.WaypointController):
85
86      search_master_searcher_id = None #variable for master searcher id
87      #2D Array to store the 'cell' objects
88      searchGrid = None
89      # Search Swarm UAV status
90      searchUAVMap = dict()
91
92      # Assume Search Grid message states BottomLeft LLA plus length and width
93      bottomLeftLLA = None
94      centerofSearchGridLLA = None
95      cols = None
96      rows = None
97
98      # default search Algo enum
99      #selectedSearchAlgo = 1
100     selectedSearchAlgo = None
101     numOfSearcher = 0
102     rowSearch = False
103     prePlannedPathPlanned = 0
104
```

```
105     # SearchSubSwarmID
106     searchSubSwarmID = 0
107
108     numOfRunsToDo = 0 # 0 means unlimited
109     numOfRunsDone = 0 #
110
111     # @param nodename: name of the ROS node in which this object exists
112     # @param ownAC: ID (int) of this aircraft
113     def __init__(self, nodename, ownAC, args):
114         wp_controller.WaypointController.__init__(self, nodename, \
115             enums.SWARM_SEARCH_CTRLR)
116         self.DBUG_PRINT = False
117         self.WARN_PRINT = False
118         # Boolean flag to determine Centralised search node
119         self.SEARCH_MASTER = False
120         self.SEARCH_STATUS = SEARCH_READY
121         self.ownID = ownAC
122         self.ExecuteMasterSearcherBehavior = False
123
124         # Used to determine altitude for wpt orders
125         self._wp_rel_alt = None
126         self._crnt_wp_id = None
127
128         # Counter used to assign safety altitude to each UAV
129         self.uavAltitudeCounter = 0
130
131         # Time when search operation begins
132         self.search_Operation_StartTime = None
133         # Time when current search runs begins
```

```
134         self.search_Run_StartTime = None
135         # Boolean when first search cell found
136         self.search_Run_firstSearch = False
137 
138         # Boolean flag to determine if any waypoint message to send
139         self.wpmsgQueued = False
140 
141         self._getWpSrvProxy = None
142         self._deactivateSrvProxy = None
143         self._swarmSearchPublisher = None
144         self._assignedSearchMessage = apmsg.SwarmSearchWaypointList()
145 
146 
147     #--------------------------------------------------
148     # Implementation of parent class virtual functions
149     #--------------------------------------------------
150 
151     # @param params: no additional parameters required by this method
152     def callbackSetup(self, params=[]):
153         self.createSubscriber("swarm_uav_states", apmsg.SwarmStateStamped, \
154                               self._process_swarm_uav_states)
155         self.createSubscriber("recv_swarm_search_waypoint", apmsg.SwarmSearchWaypointList, \
156                                 self._process_swarm_search_waypoint)
157         self.createSubscriber("swarm_search_setup", apmsg.SwarmSearchOrderStamped, \
158                                 self._process_swarmSearch_setup)
159         self.createSubscriber("status", apbrgmsg.Status, \
160                               self.sub_autopilot_status_update)
161 
162     def publisherSetup(self, params=[]):
```

```
163         self._swarmSearchPublisher = \
164             self.createPublisher("send_swarm_search_waypoint", apmsg.SwarmSearchWaypointList, 1)
165 
166     def serviceSetup(self, params=[]):
167         return
168 
169     def serviceProxySetup(self, params=[]):
170         self._deactivateSrvProxy = \
171             self.createServiceProxy("set_swarm_behavior", apsrv.SetInteger)
172         self._getWpSrvProxy = \
173             self.createServiceProxy("wp_getrange", apbrgsrv.WPGetRange)
174 
175     def runController(self):
176         if self.ExecuteMasterSearcherBehavior == True:
177             self.wpmsgQueued = False #flag to determine if any waypoint message to send
178             del self._assignedSearchMessage.waypoints[:]    # Clear current message contents
179 
180             for vehicle in self.searchUAVMap:
181                 self.log_dbug("\033[94m TimeLoop: UAV \033[96m[" + str(vehicle) + \
182                               "] \033[94m Received State [" + str(self.searchUAVMap[vehicle].
                                  receivedState) + \
183                               "] pose: \033[0m" + str(self.searchUAVMap[vehicle].pose))
184 
185             if self.SEARCH_STATUS == SEARCH_READY:
186                 if self.numOfRunsToDo == 0 or self.numOfRunsDone < self.numOfRunsToDo:
187                     self.SEARCH_STATUS = SEARCH_INGRESS
188                     self.search_Run_StartTime = time.time()
189                     self.log_dbug("\033[94mSwarm Search Master \033[96m[" + str(self.ownID) + \
190                                   "]\033[94m current search run \033[96m" + str(self.
```

```
                                    numOfRunsDone+1) + \
191                                 "\033[94m started on \033[96m" + \
192                                 str(time.asctime(time.localtime(time.time())))+"\033[0m")
193                     self.search_Run_firstSearch = False
194             elif self.SEARCH_STATUS == SEARCH_INGRESS:
195                 self._assign_ready_UAVs_to_searchGrid()
196                 self._check_timeout()
197                 # Check if any of the search uav has reached the search area. set search status
                        to active if true
198                 reached = self._assign_ingress_UAVs_to_search()
199
200                 if reached == True:
201                     self.SEARCH_STATUS = SEARCH_ACTIVE
202             elif self.SEARCH_STATUS == SEARCH_ACTIVE:
203                 self._assign_ready_UAVs_to_searchGrid()
204                 self._assign_ingress_UAVs_to_search()
205                 self._check_timeout()
206
207                 # Check if search is completed (all cells visited) set status to SEARCH_EGRESS
208                 searchComplete = self._check_search_operation()
209
210                 if searchComplete == True:
211                     self.numOfRunsDone += 1
212                     currentTime = time.time()
213                     timeDelta = currentTime - self.search_Run_StartTime
214                     self.log_dbug("\033[94mSwarm Search Master \033[96m[" + str(self.ownID) + \
215                                   "]\033[94m current search run \033[96m" + str(self.
                                      numOfRunsDone) + \
216                                   " \033[94mcompleted after \033[96m" + \
```

```
217                             "{0:.1f}".format(timeDelta) + "\033[94m secs\033[0m")
218
219                 for j in range(self.rows):
220                     for i in range(self.cols):
221                         timeDelta = self.searchGrid[i][j].visitedTimeStamp - self.
                                search_Run_StartTime
222                         self.log_dbug("\033[93m [" + str(i) + "," + str(j) + \
223                                       "] \033[94mVisit by \033[96m" + str(self.searchGrid[i][
                                           j].visitedBy) + \
224                                       "\033[94m after \033[96m" + "{0:.1f}".format(timeDelta)
                                           + \
225                                       "\033[94m secs\033[0m")
226
227                 self.SEARCH_STATUS = SEARCH_EGRESS
228
229                 # set all ingress/active uav to egress
230                 for vehicle in self.searchUAVMap:
231                     if self.searchUAVMap[vehicle].status == SEARCH_INGRESS or \
232                        self.searchUAVMap[vehicle].status == SEARCH_ACTIVE:
233                         self.searchUAVMap[vehicle].status = SEARCH_EGRESS
234                         self.searchUAVMap[vehicle].assignedTime = time.time()
235                         lla = apbrgmsg.LLA()
236                         lla.lat = self.searchUAVMap[vehicle].originalLLA[0]
237                         lla.lon = self.searchUAVMap[vehicle].originalLLA[1]
238                         lla.alt = self.searchUAVMap[vehicle].assignedAltitude
239                         if vehicle == self.search_master_searcher_id:
240                             self.publishWaypoint(lla)
241
242             elif self.SEARCH_STATUS == SEARCH_EGRESS:
```

```
243                 # check if all search slaves has reach originalLLA and set status to search ready
244                 # and clear search grid
245                 self._check_timeout()
246                 egressComplete = True
247                 for vehicle in self.searchUAVMap:
248                     if self.searchUAVMap[vehicle].status == SEARCH_EGRESS:
249                         egressComplete = False
250                         currentPose = self.searchUAVMap[vehicle].pose
251                         tgt_cell = self.searchUAVMap[vehicle].originalLLA
252
253                         if gps_distance(currentPose[0], currentPose[1], tgt_cell[0], tgt_cell[1])
                                < \
254                                    SEARCH_ARRIVAL_THRESHOLD:
255                             if vehicle != self.search_master_searcher_id:
256                                 _wpMsg = wpMsg()
257                                 _wpMsg.recipientvehicle_id = vehicle
258                                 _wpMsg.waypoint.lat = self.searchUAVMap[vehicle].originalLLA[0]
259                                 _wpMsg.waypoint.lon = self.searchUAVMap[vehicle].originalLLA[1]
260                                 _wpMsg.waypoint.alt = self.searchUAVMap[vehicle].assignedAltitude
261                                 _wpMsg.searchCell_x = 255
262                                 _wpMsg.searchCell_y = 255
263                                 self._assignedSearchMessage.waypoints.append(_wpMsg)
264                                 self.wpmsgQueued = True
265
266                             else:
267                                 self.searchUAVMap[vehicle].status = SEARCH_READY
268
269                 if egressComplete == True:
270                     self.log_dbug("\033[94mSwarm Search Master \033[96m[" + str(self.ownID) + \
```

```
271                                 "]\033[94m Master searcher has egressed and all slave searchers
                                        deactivated\033[0m")
272                     self.SEARCH_STATUS = SEARCH_READY
273                     self.searchGrid = None
274
275                     if self.numOfRunsDone == self.numOfRunsToDo:
276                         currentTime = time.time()
277                         timeDelta = currentTime - self.search_Operation_StartTime
278                         self.log_dbug("\033[94mSwarm Search Master \033[96m[" + str(self.ownID) +
                                \
279                                       "]\033[94m entire search operation completed with \033[96m"
                                           + \
280                                       str(self.numOfRunsToDo) + " \033[94mruns \033[94mafter
                                           \033[96m" + \
281                                       "{0:.3f}".format(timeDelta)+"\033[94m secs\033[0m")
282                         self.SEARCH_STATUS = SEARCH_READY
283                         self.searchGrid = None
284                         self.ExecuteMasterSearcherBehavior = False
285                         self._deactivateSrvProxy(enums.SWARM_STANDBY)
286
287                     else:
288                         self._activate_swarm_search()
289
290             if self.wpmsgQueued == True: #A new network waypoint cmd is triggered this tick
291                 self._swarmSearchPublisher.publish(self._assignedSearchMessage)
292
293
294     #--------------------------
295     # Object-specific functions
```

```
296     #-------------------------
297     def _activate_swarm_search(self, searchAlgoEnum):
298         self.log_dbug("\033[94mSwarm Search Master \033[96m[" + str(self.ownID) + \
299                       "]\033[94m generate Search Grid:\033[0m")
300         # Create the searchGrid
301         self.searchGrid = [[cell() for j in range(self.rows)] for i in range(self.cols)]
302         # populate the searchGrid
303         for j in range(self.rows):
304             for i in range(self.cols):
305                 self.searchGrid[i][j].grid = [i * SEARCH_CELL_RADIUS, j * SEARCH_CELL_RADIUS]
306                 self.searchGrid[i][j].LLA = gps_offset(self.bottomLeftLLA[0], \
307                                                        self.bottomLeftLLA[1], \
308                                                        i * SEARCH_CELL_RADIUS, \
309                                                        j * SEARCH_CELL_RADIUS)
310                 self.log_dbug("\033[93m [" + str(i) + "," + str(j) + \
311                               "] \033[36m Grid: " + str(self.searchGrid[i][j].grid) + \
312                               " \033[0mLLA: " + str(self.searchGrid[i][j].LLA))
313         self.centerofSearchGridLLA = self.searchGrid[self.cols/2][self.rows/2].LLA
314         if searchAlgoEnum == 2: # Preplanned swarm search selected
315             self.numOfSearcher = 0
316             self.rowSearch = False
317             for vehicle in self.searchUAVMap:
318                 if self.searchUAVMap[vehicle].subSwarmID == self.searchSubSwarmID:
319                     self.searchUAVMap[vehicle].preplannedPath = self.numOfSearcher
320                     self.numOfSearcher += 1
321             if self.numOfSearcher < self.rows and self.numOfSearcher < self.cols:
322                 self.log_dbug("\033[94mSwarm Search Master \033[96m["+str(self.ownID)+"]\033[94m"
                        + \
323                               " Preplanned swarm search algo requires searchers to be at least "
```

```
                                + \
324                             "the number of rows or columns of the Search Grid:\033[0m")
325             self.log_dbug("\033[94mSwarm Search Master \033[96m["+str(self.ownID)+"]\033[94m"
                    + \
326                             "Defaulting to greedy swarm search algo.\033[0m")
327         else:
328             self.prePlannedPathPlanned = 0
329             self.selectedSearchAlgo = 2
330             if self.rows > self.cols:
331                 if self.numOfSearcher >= self.rows:
332                     self.rowSearch = True
333                 else:
334                     self.rowSearch = False
335             else:
336                 if self.numOfSearcher >= self.cols:
337                     self.rowSearch = False
338                 else:
339                     self.rowSearch = True
340             if self.rowSearch == True:
341                 self.log_dbug("\033[94mSwarm Search Master \033[96m["+str(self.ownID) + \
342                                 "]\033[94m Preplanned swarm search algo activated with
                                        \033[36m" + \
343                                 str(self.rows)+" rows\033[94m search\033[0m")
344                 self.prePlannedPathPlanned = self.rows
345             else:
346                 self.log_dbug("\033[94mSwarm Search Master \033[96m["+str(self.ownID) + \
347                                 "]\033[94m Preplanned swarm search algo activated with
                                        \033[36m" + \
348                                 str(self.cols)+" columns\033[94m search\033[0m")
```

```
349                     self.prePlannedPathPlanned = self.cols
350
351
352     def _assign_ready_UAVs_to_searchGrid(self):
353         for vehicle in self.searchUAVMap:
354             if self.searchUAVMap[vehicle].status == SEARCH_READY and \
355                self.searchUAVMap[vehicle].subSwarmID == self.searchSubSwarmID:
356                 self.searchUAVMap[vehicle].status = SEARCH_INGRESS
357                 if self.searchUAVMap[vehicle].originalLLA == None:
358                     self.searchUAVMap[vehicle].originalLLA = self.searchUAVMap[vehicle].pose
359                 self.searchUAVMap[vehicle].assignedLLA = self.centerofSearchGridLLA
360                 self.searchUAVMap[vehicle].assignedCell = [self.cols/2, self.rows/2]
361                 self.searchUAVMap[vehicle].assignedTime = time.time()
362                 self.searchUAVMap[vehicle].timeoutdistance = gps_distance(self.searchUAVMap[
                        vehicle].pose[0], \
363                                                                           self.searchUAVMap[
                                                                               vehicle].pose[1], \
364                                                                           self.
                                                                               centerofSearchGridLLA
                                                                               [0], \
365                                                                           self.
                                                                               centerofSearchGridLLA
                                                                               [1])
366                 self.searchUAVMap[vehicle].timeoutTime = time.time()
367                 if vehicle == self.search_master_searcher_id:
368                     lla = apbrgmsg.LLA()
369                     lla.lat = self.centerofSearchGridLLA[0]
370                     lla.lon = self.centerofSearchGridLLA[1]
371                     lla.alt = self.searchUAVMap[vehicle].assignedAltitude
```

```
372                     self.publishWaypoint(lla)
373
374                 else:
375                     _wpMsg = wpMsg()
376                     _wpMsg.recipientvehicle_id = vehicle
377                     _wpMsg.waypoint.lat = self.centerofSearchGridLLA[0]
378                     _wpMsg.waypoint.lon = self.centerofSearchGridLLA[1]
379                     _wpMsg.waypoint.alt = self.searchUAVMap[vehicle].assignedAltitude
380                     _wpMsg.searchCell_x = self.cols/2
381                     _wpMsg.searchCell_y = self.rows/2
382                     self._assignedSearchMessage.waypoints.append(_wpMsg)
383                     self.wpmsgQueued = True
384                 self.log_dbug("\033[94m" + "Swarm Search node: Searcher \033[96m[" + \
385                               str(vehicle) + "]\033[94m ingress to search area:" + "\033[0m")
386
387
388     def _assign_ingress_UAVs_to_search(self):
389         reached = False
390         currentTime = time.time()
391         for vehicle in self.searchUAVMap:
392             if self.searchUAVMap[vehicle].status == SEARCH_INGRESS and \
393                self.searchUAVMap[vehicle].subSwarmID == self.searchSubSwarmID:
394                 currentPose = self.searchUAVMap[vehicle].pose
395                 for j in range(self.rows):
396                     for i in range(self.cols):
397                         if reached == False:
398                             tgt_cell = self.searchGrid[i][j].LLA
399                             if gps_distance(currentPose[0], currentPose[1], \
400                                             tgt_cell[0], tgt_cell[1]) < SEARCH_ARRIVAL_THRESHOLD:
```

```
401                                 reached = True
402                                 self.searchUAVMap[vehicle].status = SEARCH_ACTIVE
403                                 if self.selectedSearchAlgo == 2:
404                                     if self.searchUAVMap[vehicle].preplannedPath < self.
                                        prePlannedPathPlanned:
405                                         if self.rowSearch == True:
406                                             tgt_cell = self.searchGrid[0][self.searchUAVMap[
                                                vehicle].preplannedPath].LLA
407                                             tgt_cell2 = self.searchGrid[self.cols-1][self.
                                                searchUAVMap[vehicle].preplannedPath].LLA
408                                             if gps_distance(currentPose[0], currentPose[1],
                                                tgt_cell[0], tgt_cell[1]) \
409                                                < gps_distance(currentPose[0], currentPose[1],
                                                    tgt_cell2[0], tgt_cell2[1]):
410                                                 self.searchUAVMap[vehicle].assignedCell = [0,self
                                                    .searchUAVMap[vehicle].preplannedPath]
411                                                 self.log_dbug("\033[94m" + "Swarm Search node:
                                                    Searcher \033[96m["+ \
412                                                                 str(vehicle)+"]\033[94m assigned
                                                                    entry via: " + \
413                                                                 "\033[93m["+str(0)+","+str(self.
                                                                    searchUAVMap[vehicle].
                                                                    preplannedPath) + \
414                                                                 "]\033[0m")
415                                             else:
416                                                 tgt_cell = self.searchGrid[self.cols-1][self.
                                                    searchUAVMap[vehicle].preplannedPath].LLA
417                                                 self.searchUAVMap[vehicle].assignedCell = [self.
                                                    cols-1, self.searchUAVMap[vehicle].
```

```
                preplannedPath]
418             self.log_dbug("\033[94m" + "Swarm Search node:
                    Searcher \033[96m["+str(vehicle)+ \
419                         "]\033[94m assigned entry via: " +
                            "\033[93m["+str(self.cols-1)+ \
420                         ","+str(self.searchUAVMap[vehicle].
                            preplannedPath)+"]\033[0m")
421     else:
422         tgt_cell = self.searchGrid[self.searchUAVMap[vehicle
                ].preplannedPath][0].LLA
423         tgt_cell2 = self.searchGrid[self.searchUAVMap[vehicle
                ].preplannedPath][self.rows-1].LLA
424         if gps_distance(currentPose[0], currentPose[1],
                tgt_cell[0], tgt_cell[1]) \
425            < gps_distance(currentPose[0], currentPose[1],
                   tgt_cell2[0], tgt_cell2[1]):
426             self.searchUAVMap[vehicle].assignedCell = [self.
                    searchUAVMap[vehicle].preplannedPath,0]
427             self.log_dbug("\033[94m" + "Swarm Search node:
                    Searcher \033[96m["+ \
428                         str(vehicle)+"]\033[94m assigned
                            entry via: " + "\033[93m["+ \
429                         str(self.searchUAVMap[vehicle].
                            preplannedPath)+","+str(0)+"
                            ]\033[0m")
430         else:
431             tgt_cell = self.searchGrid[self.searchUAVMap[
                    vehicle].preplannedPath][self.rows-1].LLA
432             self.searchUAVMap[vehicle].assignedCell = [self.
```

```
                                searchUAVMap[vehicle].preplannedPath, self.
                                rows-1]
433                         self.log_dbug("\033[94m" + "Swarm Search node:
                                Searcher \033[96m["+str(vehicle)+ \
434                                     "]\033[94m assigned entry via: " +
                                        "\033[93m["+ \
435                                     str(self.searchUAVMap[vehicle].
                                        preplannedPath)+","+str(self.
                                        rows-1)+"]\033[0m")
436                 else: #terminate extra searchers
437                     self.searchUAVMap[vehicle].assignedCell = [254,254]
438                     tgt_cell = self.searchUAVMap[vehicle].originalLLA
439                     self.searchUAVMap[vehicle].status = SEARCH_EGRESS
440                     self.log_dbug("\033[94m" + "Swarm Search node: Extra
                            searcher \033[96m["+str(vehicle)+ \
441                                 "]\033[94m removed from search\033[0m")
442             else:
443                 self.searchUAVMap[vehicle].assignedCell = [i,j]
444                 self.log_dbug("\033[94m" + "Swarm Search node: Searcher
                        \033[96m["+str(vehicle)+ \
445                             "]\033[94m assigned entry via: " + "\033[93m["+
                                str(i)+","+str(j)+"]\033[0m")
446             self.searchUAVMap[vehicle].assignedTime = currentTime
447             self.searchUAVMap[vehicle].assignedLLA = tgt_cell
448             self.searchUAVMap[vehicle].timeoutdistance = \
449                 gps_distance(currentPose[0], currentPose[1], tgt_cell[0],
                        tgt_cell[1])
450             self.searchUAVMap[vehicle].timeoutTime = currentTime
451             #assign UAV to tgt_cell
```

```
452                                 if vehicle == self.search_master_searcher_id:
453                                     lla = apbrgmsg.LLA()
454                                     lla.lat = tgt_cell[0]
455                                     lla.lon = tgt_cell[1]
456                                     lla.alt = self.searchUAVMap[vehicle].assignedAltitude
457                                     self.publishWaypoint(lla)
458
459                                 else:
460                                     _wpMsg = wpMsg()
461                                     _wpMsg.recipientvehicle_id = vehicle
462                                     _wpMsg.waypoint.lat = tgt_cell[0]
463                                     _wpMsg.waypoint.lon = tgt_cell[1]
464                                     _wpMsg.waypoint.alt = self.searchUAVMap[vehicle].
                                        assignedAltitude
465                                     _wpMsg.searchCell_x = self.searchUAVMap[vehicle].assignedCell
                                        [0]
466                                     _wpMsg.searchCell_y = self.searchUAVMap[vehicle].assignedCell
                                        [1]
467                                     self._assignedSearchMessage.waypoints.append(_wpMsg)
468                                     self.wpmsgQueued = True
469         return reached
470
471
472     def _check_search_operation(self):
473         runOutofCells=False
474         currentTime = time.time()
475
476         for vehicle in self.searchUAVMap:
477             if self.searchUAVMap[vehicle].status == SEARCH_ACTIVE and \
```

```
478                 self.searchUAVMap[vehicle].subSwarmID == self.searchSubSwarmID:
479                  currentPose = self.searchUAVMap[vehicle].pose
480                  tgt_cell = self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][self.
                         searchUAVMap[vehicle].assignedCell[1]].LLA
481
482                  if gps_distance(currentPose[0], currentPose[1], tgt_cell[0], tgt_cell[1]) <
                         SEARCH_CELL_THRESHOLD:
483                      if self.search_Run_firstSearch == False:
484                          self.search_Run_firstSearch = True
485                          timeDelta = currentTime - self.search_Run_StartTime
486                          self.search_Run_StartTime = currentTime
487                          self.log_dbug("\033[94mSwarm Search Master \033[96m["+str(self.ownID)+"
                                 ]\033[94m actual search started after \033[96m"+ \
488                                          "{0:.1f}".format(timeDelta)+"\033[94m secs\033[0m")
489                      if self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][self.
                             searchUAVMap[vehicle].assignedCell[1]].visited == False:
490                          self.log_dbug("\033[94m" + "Swarm Search node: Searcher \033[96m[" + \
491                                        str(vehicle) + "]\033[92m searched \033[94mcell \033[93m" +
                                             \
492                                        str(self.searchUAVMap[vehicle].assignedCell) + "\033[0m")
493                          self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][self.
                             searchUAVMap[vehicle].assignedCell[1]].visited = True
494                          self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][self.
                             searchUAVMap[vehicle].assignedCell[1]].visitedBy = vehicle
495                          self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][self.
                             searchUAVMap[vehicle].assignedCell[1]].visitedTimeStamp = time.time()
496
497                      else:
498                          self.log_dbug("\033[94m" + "Swarm Search node: Searcher \033[96m[" + \
```

```
499                         str(vehicle) + "]\033[94m visited cell \033[93m" + \
500                         str(self.searchUAVMap[vehicle].assignedCell) + "\033[0m")
501             result = self._assign_search_UAVs_to_nextCell(vehicle)
502
503             if result[0] == SEARCH_CELL_STILL_AVAILABLE: #assign UAV to tgt_cell
504                 self.searchUAVMap[vehicle].assignedCell = [result[1],result[2]]
505                 self.searchUAVMap[vehicle].assignedTime = currentTime
506                 self.searchGrid[result[1]][result[2]].assigned = True
507                 self.searchGrid[result[1]][result[2]].assignedTo = vehicle
508                 self.searchUAVMap[vehicle].assignedLLA = self.searchGrid[result[1]][
                        result[2]].LLA
509                 self.searchUAVMap[vehicle].timeoutdistance = \
510                     gps_distance(currentPose[0], currentPose[1], \
511                                  self.searchGrid[result[1]][result[2]].LLA[0], \
512                                  self.searchGrid[result[1]][result[2]].LLA[1])
513                 self.searchUAVMap[vehicle].timeoutTime = currentTime
514                 self.log_dbug("\033[94m" + "Swarm Search node: Searcher \033[96m[" + \
515                         str(vehicle) + "]\033[94m assigned cell \033[93m" + \
516                         str(self.searchUAVMap[vehicle].assignedCell) + "\033[0m")
517
518                 if vehicle == self.search_master_searcher_id:
519                     lla = apbrgmsg.LLA()
520                     lla.lat = self.searchGrid[result[1]][result[2]].LLA[0]
521                     lla.lon = self.searchGrid[result[1]][result[2]].LLA[1]
522                     lla.alt = self.searchUAVMap[vehicle].assignedAltitude
523                     self.publishWaypoint(lla)
524
525                 else:
526                     _wpMsg = wpMsg()
```

```
527                         _wpMsg.recipientvehicle_id = vehicle
528                         _wpMsg.waypoint.lat = self.searchGrid[result[1]][result[2]].LLA[0]
529                         _wpMsg.waypoint.lon = self.searchGrid[result[1]][result[2]].LLA[1]
530                         _wpMsg.waypoint.alt = self.searchUAVMap[vehicle].assignedAltitude
531                         _wpMsg.searchCell_x = result[1]
532                         _wpMsg.searchCell_y = result[2]
533                         self._assignedSearchMessage.waypoints.append(_wpMsg)
534                         self.wpmsgQueued = True
535
536                 else:
537                     if result[0] == SEARCH_CELL_FULLY_VISITED:
538                         runOutofCells = True
539                     self.searchUAVMap[vehicle].status = SEARCH_EGRESS
540                     self.searchUAVMap[vehicle].assignedTime = currentTime
541                     self.searchUAVMap[vehicle].assignedLLA = self.searchUAVMap[vehicle].
                        originalLLA
542                     self.searchUAVMap[vehicle].timeoutdistance = \
543                         gps_distance(currentPose[0], currentPose[1], \
544                                      self.searchUAVMap[vehicle].originalLLA[0], \
545                                      self.searchUAVMap[vehicle].originalLLA[1])
546                     self.searchUAVMap[vehicle].timeoutTime = currentTime
547                     self.log_dbug("\033[94m" + "Swarm Search node: Searcher \033[96m[" + \
548                                   str(vehicle) + "]\033[94m returning home\033[0m")
549
550                     if vehicle == self.search_master_searcher_id:
551                         lla = apbrgmsg.LLA()
552                         lla.lat = self.searchUAVMap[vehicle].originalLLA[0]
553                         lla.lon = self.searchUAVMap[vehicle].originalLLA[1]
554                         lla.alt = self.searchUAVMap[vehicle].assignedAltitude
```

```
555                             self.publishWaypoint(lla)
556
557                         else:
558                             _wpMsg = wpMsg()
559                             _wpMsg.recipientvehicle_id = vehicle
560                             _wpMsg.waypoint.lat = self.searchUAVMap[vehicle].originalLLA[0]
561                             _wpMsg.waypoint.lon = self.searchUAVMap[vehicle].originalLLA[1]
562                             _wpMsg.waypoint.alt = self.searchUAVMap[vehicle].assignedAltitude
563                             _wpMsg.searchCell_x = 254
564                             _wpMsg.searchCell_y = 254
565                             self._assignedSearchMessage.waypoints.append(_wpMsg)
566                             self.wpmsgQueued = True
567         return runOutofCells
568
569
570     def _assign_search_UAVs_to_nextCell(self, vehicleID):
571         # assign new unvisited cell if possible
572         cellResult = SEARCH_CELL_FULLY_VISITED
573         currentCol = self.searchUAVMap[vehicleID].assignedCell[0]
574         currentRow = self.searchUAVMap[vehicleID].assignedCell[1]
575         tgt_cell = [0,0]
576         distance = SEARCH_MAX_THRESHOLD
577         currentPose = self.searchUAVMap[vehicleID].pose
578         if self.selectedSearchAlgo == 2:
579             if self.rowSearch == True: # find next closest cell in the same row
580                 for i in range(self.cols):
581                     if self.searchGrid[i][currentRow].visited == False:
582                         if cellResult == SEARCH_CELL_FULLY_VISITED:
583                             cellResult = SEARCH_CELL_FULLY_ASSIGNED
```

```
584                         if self.searchGrid[i][currentRow].assigned == False:
585                             cellResult = SEARCH_CELL_STILL_AVAILABLE
586                             cell = self.searchGrid[i][currentRow].LLA
587                             tempDist=gps_distance(currentPose[0], currentPose[1], cell[0], cell
                                    [1])
588                             if tempDist < distance:
589                                 distance = tempDist
590                                 tgt_cell = [i,currentRow]
591             else:
592                 for j in range(self.rows): # find next closest cell in the same col
593                     if self.searchGrid[currentCol][j].visited == False:
594                         if cellResult == SEARCH_CELL_FULLY_VISITED:
595                             cellResult = SEARCH_CELL_FULLY_ASSIGNED
596                         if self.searchGrid[currentCol][j].assigned == False:
597                             cellResult = SEARCH_CELL_STILL_AVAILABLE
598                             cell = self.searchGrid[currentCol][j].LLA
599                             tempDist=gps_distance(currentPose[0], currentPose[1], cell[0], cell
                                    [1])
600                             if tempDist < distance:
601                                 distance = tempDist
602                                 tgt_cell = [currentCol,j]
603             if cellResult == SEARCH_CELL_FULLY_VISITED: # To check if the whole search grid is
                    fully visited
604                 for j in range(self.rows):
605                     for i in range(self.cols):
606                         if self.searchGrid[i][j].visited == False:
607                             cellResult = SEARCH_CELL_FULLY_ASSIGNED
608         else:
609             for j in range(self.rows):
```

```
610                 for i in range(self.cols):
611                     if self.searchGrid[i][j].visited == False:
612                         if cellResult == SEARCH_CELL_FULLY_VISITED:
613                             cellResult = SEARCH_CELL_FULLY_ASSIGNED
614                         if self.searchGrid[i][j].assigned == False:
615                             cellResult = SEARCH_CELL_STILL_AVAILABLE
616                             cell = self.searchGrid[i][j].LLA
617                             tempDist=gps_distance(currentPose[0], currentPose[1], cell[0], cell
                                [1])
618                             if tempDist < distance:
619                                 distance = tempDist
620                                 tgt_cell = [i,j]
621         if cellResult == SEARCH_CELL_FULLY_VISITED:
622             return [SEARCH_CELL_FULLY_VISITED,0,0]
623 
624         else:
625             return [cellResult, tgt_cell[0], tgt_cell[1]]
626 
627 
628     def _check_timeout(self):
629         # scan through all UAVs, if any did not get closer in the last interval, resend LLA
630         currentTime = time.time()
631         for vehicle in self.searchUAVMap:
632             #any states that has assigned waypoints
633             if self.searchUAVMap[vehicle].status in (SEARCH_INGRESS, SEARCH_ACTIVE, SEARCH_EGRESS
                ):
634                 # timeout interval up
635                 if self.searchUAVMap[vehicle].timeoutTime + SEARCH_TIMEOUT_INTERVAL < currentTime
                    :
```

```
636                     distFromTgt = gps_distance(self.searchUAVMap[vehicle].pose[0], \
637                                                self.searchUAVMap[vehicle].pose[1], \
638                                                self.searchUAVMap[vehicle].assignedLLA[0], \
639                                                self.searchUAVMap[vehicle].assignedLLA[1])
640
641                     #uav is closer since last check. Update latest distance and time
642                     if distFromTgt < self.searchUAVMap[vehicle].timeoutdistance:
643                         if self.searchUAVMap[vehicle].status == SEARCH_EGRESS:
644                             self.searchUAVMap[vehicle].timeoutdistance = \
645                                 distFromTgt + SEARCH_TIMEOUT_LOITERDISTANCEBUFFER
646
647                         else:
648                             self.searchUAVMap[vehicle].timeoutdistance = distFromTgt
649                         self.searchUAVMap[vehicle].timeoutTime = currentTime
650                         self.searchUAVMap[vehicle].timeoutCounter = 0
651
652                     else:
653                         # resend assigned lla
654                         if self.searchUAVMap[vehicle].timeoutCounter < SEARCH_TIMEOUT_STRIKES:
655                             self.searchUAVMap[vehicle].timeoutCounter += 1
656                             self.log_dbug("\033[94mSwarm Search node: \033[91mTIMEOUT " + \
657                                           str(self.searchUAVMap[vehicle].timeoutCounter) + \
658                                           "\033[94m for UAV \033[96m[" + str(vehicle) + \
659                                           "] \033[94mResend LLA... \033[0m")
660                             self.searchUAVMap[vehicle].timeoutTime = currentTime
661
662                             if vehicle == self.search_master_searcher_id:
663                                 lla = apbrgmsg.LLA()
664                                 lla.lat = self.searchUAVMap[vehicle].assignedLLA[0]
```

```
665                     lla.lon = self.searchUAVMap[vehicle].assignedLLA[1]
666                     lla.alt = self.searchUAVMap[vehicle].assignedAltitude
667                     self.publishWaypoint(lla)
668
669                 else:
670                     _wpMsg = wpMsg()
671                     _wpMsg.recipientvehicle_id = vehicle
672                     _wpMsg.waypoint.lat = self.searchUAVMap[vehicle].assignedLLA[0]
673                     _wpMsg.waypoint.lon = self.searchUAVMap[vehicle].assignedLLA[1]
674                     _wpMsg.waypoint.alt = self.searchUAVMap[vehicle].assignedAltitude
675                     _wpMsg.searchCell_x = self.searchUAVMap[vehicle].assignedCell[0]
676                     _wpMsg.searchCell_y = self.searchUAVMap[vehicle].assignedCell[1]
677                     self._assignedSearchMessage.waypoints.append(_wpMsg)
678                     self.wpmsgQueued = True
679
680             else: # UAV not getting closer after all the timeout strikes, free
                    assigned cell for others and place uav out of search operation
681                 self.log_dbug("\033[94mSwarm Search node: UAV \033[96m[" + str(
                        vehicle) + \
682                               "] \033[91m STRIKEOUT! \033[0m")
683                 if self.searchUAVMap[vehicle].status == SEARCH_ACTIVE:
684                     # see if another UAV active. if so, contiune so  they can take
                            over the released cell
685                     if self._available_activeUAVHandover(vehicle) == True:
686                         self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][
                                self.searchUAVMap[vehicle].assignedCell[1]].assigned =
                                False
687                         self.searchGrid[self.searchUAVMap[vehicle].assignedCell[0]][
                                self.searchUAVMap[vehicle].assignedCell[1]].assignedTo =
```

```
                None
688             self.log_dbug("\033[94mSearch cell \033[92m" + \
689                           str(self.searchUAVMap[vehicle].assignedCell) +
                                \
690                           " \033[94m released for reassignment\033[0m")
691         else:
692             egressUAVID = \
693                 self._closest_egressUAVHandover(self.searchUAVMap[vehicle
                        ].assignedCell[0], \
694                                                 self.searchUAVMap[vehicle
                                                        ].assignedCell[1])
695
696             if egressUAVID != 0: #Found egress UAV to take over
697                 self.searchUAVMap[egressUAVID].status = SEARCH_ACTIVE
698                 self.searchUAVMap[egressUAVID].assignedCell = \
699                     self.searchUAVMap[vehicle].assignedCell
700                 self.searchUAVMap[egressUAVID].assignedTime = currentTime
701                 self.searchGrid[self.searchUAVMap[vehicle].assignedCell
                        [0]][self.searchUAVMap[vehicle].assignedCell[1]].
                        assigned = True
702                 self.searchGrid[self.searchUAVMap[vehicle].assignedCell
                        [0]][self.searchUAVMap[vehicle].assignedCell[1]].
                        assignedTo = egressUAVID
703                 self.searchUAVMap[egressUAVID].assignedLLA = self.
                        searchGrid[self.searchUAVMap[vehicle].assignedCell
                        [0]][self.searchUAVMap[vehicle].assignedCell[1]].LLA
704                 self.searchUAVMap[egressUAVID].timeoutdistance = \
705                     gps_distance(self.searchUAVMap[egressUAVID].pose[0],
                            \
```

```
706                                 self.searchUAVMap[egressUAVID].pose[1],
                                        \
707                                 self.searchUAVMap[egressUAVID].
                                        assignedLLA[0], \
708                                 self.searchUAVMap[egressUAVID].
                                        assignedLLA[1])
709                         self.searchUAVMap[egressUAVID].timeoutTime = currentTime
710                         self.log_dbug("\033[94m" + "Swarm Search node: Egressed
                                Searcher \033[96m[" + \
711                                     str(egressUAVID) + \
712                                     "]\033[94m assigned to take over strikedout
                                         \033[96m[" + \
713                                     str(vehicle)+"]\033[94m assigned cell
                                        \033[93m" + \
714                                     str(self.searchUAVMap[vehicle].assignedCell
                                        ) + "\033[0m")
715
716                         if egressUAVID == self.search_master_searcher_id:
717                             lla = apbrgmsg.LLA()
718                             lla.lat = self.searchUAVMap[egressUAVID].assignedLLA
                                    [0]
719                             lla.lon = self.searchUAVMap[egressUAVID].assignedLLA
                                    [1]
720                             lla.alt = self.searchUAVMap[egressUAVID].
                                    assignedAltitude
721                             self.publishWaypoint(lla)
722
723                         else:
724                             _wpMsg = wpMsg()
```

```
725                                         _wpMsg.recipientvehicle_id = egressUAVID
726                                         _wpMsg.waypoint.lat = self.searchUAVMap[egressUAVID].
                                                assignedLLA[0]
727                                         _wpMsg.waypoint.lon = self.searchUAVMap[egressUAVID].
                                                assignedLLA[1]
728                                         _wpMsg.waypoint.alt = self.searchUAVMap[egressUAVID].
                                                assignedAltitude
729                                         _wpMsg.searchCell_x = self.searchUAVMap[egressUAVID].
                                                assignedCell[0]
730                                         _wpMsg.searchCell_y = self.searchUAVMap[egressUAVID].
                                                assignedCell[1]
731                                         self._assignedSearchMessage.waypoints.append(_wpMsg)
732                                         self.wpmsgQueued = True
733
734                                 else:
735                                     self.log_dbug("\033[94mSearch \033[91m FAILED! \033[94m
                                            cell \033[92m" + \
736                                                 str(self.searchUAVMap[vehicle].assignedCell
                                                        ) + \
737                                                 " \033[94m released for assignment with no
                                                        UAV to take over\033[0m")
738                             self.searchUAVMap[vehicle].status = SEARCH_FAULT
739
740
741     def _available_activeUAVHandover(self, faultyUAVID):
742         available = False
743         for vehicle in self.searchUAVMap:
744             if self.searchUAVMap[vehicle].status in (SEARCH_INGRESS, SEARCH_ACTIVE):
745                 if vehicle != faultyUAVID:
```

```
746                     available = True
747         return available
748
749
750     def _closest_egressUAVHandover(self, cell_x, cell_y):
751         closestUAV = 0
752         closestDistance = SEARCH_MAX_THRESHOLD
753         tgtLLA = self.searchGrid[cell_x][cell_y].LLA
754         for vehicle in self.searchUAVMap:
755             if self.searchUAVMap[vehicle].status == SEARCH_EGRESS:
756                 distFromTgt = gps_distance(self.searchUAVMap[vehicle].pose[0], \
757                                            self.searchUAVMap[vehicle].pose[1], \
758                                            tgtLLA[0], tgtLLA[0])
759                 if distFromTgt < closestDistance:
760                     closestDistance = distFromTgt
761                     closestUAV = vehicle
762         return closestUAV
763
764
765     #-----------------------------------------------
766     # ROS Subscriber callbacks -for this object
767     #-----------------------------------------------
768     # Handle incoming swarm_uav_states messages
769     # @param swarmMsg: message containing swarm data (SwarmStateStamped)
770     def _process_swarm_uav_states(self, swarmMsg):
771         for vehicle in swarmMsg.swarm:
772             if vehicle.vehicle_id in self.searchUAVMap:
773                 self.searchUAVMap[vehicle.vehicle_id].pose = \
774                     [vehicle.state.pose.pose.position.lat, vehicle.state.pose.pose.position.lon]
```

```
775                 self.searchUAVMap[vehicle.vehicle_id].subSwarmID = vehicle.subswarm_id
776
777                 if vehicle.swarm_state != self.searchUAVMap[vehicle.vehicle_id].receivedState:
778                     if self.ExecuteMasterSearcherBehavior == True:
779                         if self.searchUAVMap[vehicle.vehicle_id].receivedState == \
780                                 SEARCH_SWARM_SEARCH_ACTIVE_STATE and \
781                             vehicle.swarm_state == SEARCH_SWARM_SEARCH_READY_STATE:
782                             self.log_dbug("\033[92mUAV \033[96m[" + str(vehicle.vehicle_id) + \
783                                           "] \033[94mterminated swarm behaviour. \033[0m")
784                             self.searchUAVMap[vehicle.vehicle_id].status = SEARCH_FAULT
785
786                             if self.ownID == vehicle.vehicle_id:
787                                 currentTime = time.time()
788                                 timeDelta = currentTime - self.search_Operation_StartTime
789                                 self.log_dbug("\033[94mSearch Operation \033[91mSUSPENDED \033[94
                                     mafter \033[96m" + \
790                                               "{0:.3f}".format(timeDelta) + "\033[94m secs\033[0m
                                                   ")
791
792                                 for j in range(self.rows):
793                                     for i in range(self.cols):
794                                         if self.searchGrid[i][j].visited == False:
795                                             self.log_dbug("\033[93m [" + str(i) + "," + str(j) +
                                                 \
796                                                           "] \033[94mwas not visited.\033[0m")
797
798                                         else:
799                                             timeDelta = self.searchGrid[i][j].visitedTimeStamp -
                                                 \
```

```
800                                                 self.search_Run_StartTime
801                                             self.log_dbug("\033[93m [" + str(i) + "," + str(j) +
                                                    \
802                                                     "] \033[94mVisit by \033[96m" + \
803                                                     str(self.searchGrid[i][j].visitedBy) +
                                                        \
804                                                     "\033[94m after \033[96m" + "{0:.1f}".
                                                        format(timeDelta) + \
805                                                     "\033[94m secs\033[0m")
806
807                                 self.SEARCH_STATUS = SEARCH_READY
808                                 self.searchGrid = None
809                                 self.ExecuteMasterSearcherBehavior = False
810                                 self._deactivateSrvProxy(enums.SWARM_STANDBY)
811
812                     self.searchUAVMap[vehicle.vehicle_id].receivedState = vehicle.swarm_state
813                     if self.searchUAVMap[vehicle.vehicle_id].receivedState ==
                            SEARCH_SWARM_SEARCH_ACTIVE_STATE:
814                         # When swarm behaviour just turn active, the first local state is search
                                ready
815                         self.searchUAVMap[vehicle.vehicle_id].status = SEARCH_READY
816
817                 else:
818                     self.searchUAVMap[vehicle.vehicle_id] = searchUAVdata()
819                     self.searchUAVMap[vehicle.vehicle_id].pose = \
820                         [vehicle.state.pose.pose.position.lat, vehicle.state.pose.pose.position.lon]
821                     self.searchUAVMap[vehicle.vehicle_id].subSwarmID = vehicle.subswarm_id
822                     self.searchUAVMap[vehicle.vehicle_id].assignedAltitude = \
823                         SEARCH_ALTITUDE + (self.uavAltitudeCounter * SEARCH_SAFETY_ALTITUDE_INTERVAL)
```

```
824                 self.uavAltitudeCounter += 1
825
826
827     def _process_swarm_search_waypoint(self, swarmSearchWP):
828         # Check if own UAV supposed to recepient of this message
829         for waypointMsg in swarmSearchWP.waypoints:
830             if self.ownID == waypointMsg.recipientvehicle_id:
831                 if waypointMsg.searchCell_x >= 254 and waypointMsg.searchCell_y >= 254:
832                     self._deactivateSrvProxy(enums.SWARM_STANDBY)
833                     self.log_dbug("\033[94m" + "Swarm Searcher \033[96m[" + str(self.ownID)+ \
834                                   "]\033[94m proceeding to deactivate\033[0m")
835
836                 else:
837                     lla = apbrgmsg.LLA()
838                     lla.lat = waypointMsg.waypoint.lat
839                     lla.lon = waypointMsg.waypoint.lon
840                     lla.alt = self.searchUAVMap[self.ownID].assignedAltitude
841                     self.publishWaypoint(lla)
842                     self.log_dbug("\033[94m" + "Swarm Searcher \033[96m[" + str(self.ownID) + \
843                                   "]\033[94m proceeding to assigned cell \033[93m[" + \
844                                   str(waypointMsg.searchCell_x) + ", " + \
845                                   str(waypointMsg.searchCell_y) + "]\033[0m")
846
847             else:
848                 if waypointMsg.searchCell_x >= 254 and waypointMsg.searchCell_y >= 254:
849                     self.log_dbug("\033[94m" + "Swarm Search \033[96m[" + str(self.ownID) + \
850                                   "]\033[94m node: Received network wp cmd for Slave Searcher
                                          \033[96m[" + \
851                                   str(waypointMsg.recipientvehicle_id) + "]\033[94m to deactivate
```

```
                                        \033[0m")
852
853
854     def _process_swarmSearch_setup(self, swarmSearchSetup):
855         try:
856             if self._crnt_wp_id is None:
857                 raise ValueError('Autopilot status has not been received yet')
858             self._wp_rel_alt = self._getWpSrvProxy(self._crnt_wp_id, self._crnt_wp_id).points[0].
                    z
859             self.search_master_searcher_id = swarmSearchSetup.order.masterSearcherID
860             if self.ownID == self.search_master_searcher_id:
861                 self.log_dbug("\033[94mSwarm Search Master \033[96m[" + str(self.ownID) + \
862                               "]\033[94m received command from swarm manager \033[96m" + \
863                               str(swarmSearchSetup) + "\033[0m")
864
865                 if self.ExecuteMasterSearcherBehavior == False:
866                     self.search_Operation_StartTime = time.time()
867                     self.bottomLeftLLA = [swarmSearchSetup.order.lat, swarmSearchSetup.order.lon]
868                     self.selectedSearchAlgo = 1 # set 1 (Greedy) as default
869                     tempVar = swarmSearchSetup.order.searchAreaLength / SEARCH_CELL_RADIUS
870                     self.cols = int(math.ceil(tempVar))
871                     tempVar = swarmSearchSetup.order.searchAreaWidth / SEARCH_CELL_RADIUS
872                     self.rows = int(math.ceil(tempVar))
873                     self.numOfRunsToDo = 1
874                     self.numOfRunsDone = 0
875                     self.searchSubSwarmID = self.searchUAVMap[self.ownID].subSwarmID
876                     self.ExecuteMasterSearcherBehavior = True
877                     self.SEARCH_STATUS = SEARCH_READY
878                     self.searchGrid = None
```

```
879                 self.log_dbug("\033[94mSwarm Search Master \033[96m[" + str(self.ownID) + \
880                               "]\033[94m search operation with \033[96m" + str(self.
                                      numOfRunsToDo) + \
881                               "\033[94m run(s) started on \033[96m" + \
882                               str(time.asctime(time.localtime(time.time()))) + "\033[0m")
883
884                 for vehicle in self.searchUAVMap:
885                     self.searchUAVMap[vehicle].assignedAltitude = self._wp_rel_alt
886                 self.log_dbug("\033[94mSwarm Search Master \033[96m[" + str(self.ownID) + \
887                               "] \033[94massigned relative altitude for search \033[0m" + \
888                               str(self.searchUAVMap[self.ownID].assignedAltitude))
889                 self._activate_swarm_search(swarmSearchSetup.order.searchAlgoEnum)
890                 self.log_dbug("\033[94mSwarm Search Master \033[96m[" + str(self.ownID) + \
891                               "] \033[94musing swarm search algorithm \033[0m" + str(self.
                                      selectedSearchAlgo))
892                 self.set_ready_state(True)
893
894             else:
895                 self.log_dbug("\033[94mSwarm Search Master \033[96m[" + str(self.ownID) + \
896                               "]\033[94m ignored command from swarm manager as search
                                      operation is underway. Suspend operation first before
                                      assigning any new operation\033[0m")
897
898         else:
899             self.ExecuteMasterSearcherBehavior = False
900             self.searchUAVMap[self.ownID].assignedAltitude = self._wp_rel_alt
901             self.log_dbug("\033[94mSwarm Search node: Searcher \033[96m[" + str(self.ownID) +
                    \
902                           "] \033[94massigned relative altitude for search \033[0m" + \
```

```
903                                 str(self.searchUAVMap[self.ownID].assignedAltitude))
904
905             self.set_ready_state(True)
906             return True
907
908         except Exception as ex:
909             self.log_warn("Failed to initialize swarm search: " + str(ex))
910             self.set_ready_state(False)
911             return False
912
913
914     def sub_autopilot_status_update(self, msg):
915         self._crnt_wp_id = msg.mis_cur
916
917 #-----------------------------------------------------------------------
918 # Main code
919
920 if __name__ == '__main__':
921     args = rospy.myargv(argv=sys.argv)
922     searcher = SwarmSearcher("swarm_searcher",rospy.get_param("aircraft_id"),args)
923     searcher.runAsNode(10, [], [], [])
```

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] C. Anderson. (2012, June 22). How I accidentally kickstarted the domestic drone boom. *Wired*. [Online]. Available: http://www.wired.com/2012/06/ff_drones/

[2] K. Sayler. (2015, June 10). *A World of Proliferated Drones: A Technology Primer*. Center for a New American Security. [Online]. Available: http://www.cnas.org/world-of-proliferated-drones-technology-primer#.VdXHzJfGqEw

[3] The UAV.com. (n.d.). Information about UAVs, UAS, UAVS. [Online]. Available: http://www.theuav.com. Accessed July 22, 2015.

[4] A. Levin. (2014, Dec. 17). Santa delivering drones for Christmas amid rising safety concern. [Online]. Available: http://www.bloomberg.com/news/articles/2014-12-17/santa-delivering-drones-for-christmas-amid-rising-safety-concern

[5] C. Arthur. (2012, Jan. 24). The history of smartphones: Timeline. *Guardian*. [Online]. Available: http://www.theguardian.com/technology/2012/jan/24/smartphones-timeline

[6] ARM. (n.d.). ARM processor architecture. [Online]. Available: http://www.arm.com/products/processors/instruction-set-architectures/index.php. Accessed July 22, 2015.

[7] T. P. Morgan. (2011, Feb. 11). ARM holdings eager for PC and server expansion. [Online]. Available: http://www.theregister.co.uk/2011/02/01/arm_holdings_q4_2010_numbers/

[8] Nick T. (2014, July 6). Did you know how many different kinds of sensors go inside a smartphone? [Online]. Available: http://www.phonearena.com/news/Did-you-know-how-many-different-kinds-of-sensors-go-inside-a-smartphone_id57885

[9] 3DR. (n.d.). 3DR Pixhawk. [Online]. Available: https://store.3drobotics.com/products/3dr-pixhawk. Accessed July 22, 2015.

[10] Spektrum. (n.d.). DSMX® technology. [Online]. Available: http://www.spektrumrc.com/Technology/DSMX.aspx. Accessed July 25, 2015.

[11] G. McCray. (n.d.). Batteries for UAV. [Online]. Available: http://dronesarefun.com/BatteriesForUAV.html. Accessed July 23, 2015.

[12] Amazon.com. (n.d.). DJI Phantom 2 Vision Quadcopter with integrated FPV camcorder. [Online]. Available: http://www.amazon.com/DJI-Quadcopter-Integrated-Discontinued-Manufacturer/dp/B00FW78710/ref=sr_1_4?ie=UTF8&qid=1438116990&sr=8-4&keywords=dji+phantom. Accessed July 28, 2015.

[13] A. Bürkle, F. Segor, and M. Kollmann, "Towards autonomous micro UAV swarms," *J. Intelligent Robotic Syst.*, vol. 61, no. 1-4, pp. 339–353, 2011.

[14] Y. Mulgaonkar, "Automated recharging for persistence missions with multiple micro aerial vehicles," Ph.D. dissertation, Univ. of Pennsylvania, Philadelphia, 2012.

[15] E. W. Frew and T. X. Brown, "Airborne communication networks for small unmanned aircraft systems," *Proceedings of the IEEE*, vol. 96, no. 12, pp. 2008–2027, 2008.

[16] J. Welsby and C. Melhuish, "Autonomous minimalist following in three dimensions: A study with small-scale dirigibles," in *Proceedings of Towards Intelligent Mobile Robots*, 2001.

[17] S. Hauert *et al.*, "Reynolds flocking in reality with fixed-wing robots: Communication range vs. maximum turning rate," *Proc. IEEE/RSJ Int. Conf. Intelligent Robots Syst.*, pp. 5015–5020, 2011.

[18] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, vol. 35, no. 4, pp. 287–300, 2013.

[19] Vicon. (2015). T-Series. [Online]. Available: http://www.vicon.com/products/camera-systems/t-series. Accessed Aug. 3, 2015.

[20] G. Vásárhelyi *et al.*, "Outdoor flocking and formation flight with autonomous aerial robots," *Proc. IEEE/RSJ Int. Conf. Intelligent Robots Syst.*, pp. 3866–3873, 2014.

[21] L. Hunsaker. (2015, August 31). ARSENL reaches its ultimate goal of 50 autonomous UAVs in flight. [Online]. Available: http://www.navy.mil/submit/display.asp?story_id=90863

[22] Hoffmann *et al.*, "Precision flight control for a multi-vehicle quadrotor helicopter testbed," *Control Engineering Practice*, vol. 19, no. 9, pp. 1023–1036, 2011.

[23] Turpin *et al.*, "Decentralized formation control with variable shapes for aerial robots," *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 23–30, 2012.

[24] Stirling *et al.*, "Indoor navigation with a swarm of flying robots," *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, pp. 4641–4647, 2012.

[25] Quintero *et al.*, "Flocking with fixed-wing uavs for distributed sensing: A stochastic optimal control approach," *Proc. American Control Conference (ACC)*, pp. 2025–2031, 2013.

[26] J. Eagle, "Search path and coverage models," class notes for Search & Detection Theory (OA3602), Dept. of Operations Research, Naval Postgraduate School, Monterey, CA, fall 2013.

[27] A. R. Washburn, *Search & Detection,* 4th ed. Catonsville, MD: Institute for Operations Research and the Management Sciences, 2002.

[28] ODROID. (n.d.). ODROID-U3. [Online]. Available: http://www.hardkernel.com/main/products/prdt_info.php?g_code=g138745696275. Accessed July 22, 2015.

[29] JSBSim. (n.d.). Sourceforge web portal. [Online]. Available: http://jsbsim.sourceforge.net/. Accessed July 10, 2015.

[30] M. Fularz, M. Kraft, A. Schmidt, and A. Kasiński, "The architecture of an embedded smart camera for intelligent inspection and surveillance," *Progress in Automation, Robotics and Measuring Techniques*, vol. 350, pp. 43–52, 2015.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California